

RECEIVED  
CENTRAL FAX CENTER

AUG 10 2005

Docket No.: 42390P5943C

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE  
BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:

Mohammad A. Abdallah et al.

Application No. 10/005,728

Filed: November 26, 2001

For: METHOD AND APPARATUS FOR  
COMPUTING A PACKED SUM OF  
ABSOLUTE DIFFERENCES

Examiner: Richard Ellis

Art Unit: 2183

CERTIFICATE OF TRANSMISSIONI hereby certify that this correspondence is being facsimile  
transmitted to the United States Patent and Trademark  
Office, Fax No. (531) 273-8300

on

8-10-05

Date

Lawrence M. Mennemeler

APPELLANT'S BRIEF UNDER 37 CFR § 41.37  
IN SUPPORT OF APPELLANT'S APPEAL TO THE BOARD OF PATENT  
APPEALS AND INTERFERENCESMail Stop Appeal Brief-Patents  
Commissioner of Patents  
PO Box 1450  
Alexandria, VA 22313-1450

Dear Sir:

Appellant hereby submits this Brief in support of an appeal from a non-final decision of the Examiner, in the above-referenced case. Appellant respectfully requests consideration of this appeal by the Board of Patent Appeals and Interference for allowance of the above-referenced patent application.

BEST AVAILABLE COPY

## TABLE OF CONTENTS

I.	REAL PARTY IN INTEREST.....	3
II.	RELATED APPEALS AND INTERFERENCES.....	3
III.	STATUS OF THE CLAIMS.....	3
IV.	STATUS OF AMENDMENTS.....	4
V.	SUMMARY OF CLAIMED SUBJECT MATTER.....	5
VI.	GROUND OF REJECTION TO BE REVIEWED ON APPEAL.....	11
VII.	ARGUMENT.....	11
VIII.	CLAIMS APPENDIX.....	58
IX.	EVIDENCE APPENDIX.....	65
X.	RELATED PROCEEDINGS APPENDIX.....	185

I. Real Party in Interest

The real party in interest in the present appeal is Intel Corporation of Santa Clara, California, the assignee of the present application.

II. Related Appeals and Interferences

There are no related appeals or interferences to appellant's knowledge that would have a bearing on any decision of the Board of Patent Appeals and Interferences.

III. Status of the Claims (independent claims shown in bold)

Claims 1-7, 8-15, 19-20, 25, 32 and 38 are canceled.

Claims 17 and 26-38 stand rejected under 35 USC § 112, second paragraph, as allegedly being indefinite.

Claims 16-18, 26-31, 33-37 and 39-42 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (Sidwell) in view of in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun).

Claims 21-22, 23-24, 33-34 and 43-44 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (Sidwell) in view of in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun) and further in view of US Patent 5,721,697 (Lee).

Non-final rejection of claims 16-18, 21-22, 23-24, 26-31, 33-37 and 39-44 is being appealed.

#### IV. Status of Amendments

A preliminary amendment, submitted by appellant on 11/6/2001 was entered. An official response to a first Office Action mailed 8/19/2003 was submitted by appellant on 1/20/2004 and was entered. A Final Office Action was mailed on 4/9/2004. Appellant responded with an amendment and official response after final on 6/9/2004, which was entered and an Advisory Action was mailed 7/9/2004. An RCE and official response, which was not accepted, were submitted by appellant on 10/11/2004. A Notice of Non-Compliant Amendment was mailed 10/25/2004. Appellant responded by submitting a corrected official response on 11/5/2004, which was not accepted. A second Notice of Non-Compliant Amendment was mailed 12/9/2004. Appellant submitted a second corrected official response on 12/20/2004, which was entered. A Non-final Office Action was mailed on 1/10/2005. A Notice of Appeal was transmitted on 6/10/2005, and an appeal ensued. Another amendment is being submitted, under 37 CFR § 41.33 and concurrent with the present appeal brief.

Accordingly, the claims stand as of the concurrently submitted amendment of 8/10/2005, and are reproduced in clean form in the Claims Appendix.



V. Summary of Claimed Subject Matter

Appellant's disclosure describes methods and apparatus for computing multiple absolute differences from packed data and summing the multiple absolute differences together to produce a result using an execution unit that also performs multiple multiply-add operations. According to one embodiment, a processor includes a decode unit to decode a packed sum of absolute differences (PSAD) instruction having an opcode format to identify a set of packed data. The decoder initiates a first set of operations responsive to decoding the PSAD instruction. An execution unit performs a first operation of the first set of operations initiated by the decode unit. According to another embodiment, the processor also executes instructions of the PENTIUM® microprocessor instruction set.

According to another embodiment, the first set of operations comprises a packed subtract and write carry (PSUBWC) operation; a packed absolute value and read carry (PABSRC) operation; and a packed add horizontal (PADDH) operation.

According to another embodiment, performing the first operation causes the execution unit to produce a first plurality of partial products in a multiplier having a plurality of partial product selectors, to insert elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions, and to add the first plurality of elements together to produce a sum of the first plurality of elements.

According to another embodiment, the decode unit also decodes a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data and initiates a second set of operations responsive to decoding the PMAD instruction. The execution unit also performs a second operation of the second set of operations initiated by the decode unit.

According to another embodiment, performing the second operation causes the execution unit to produce a second plurality of partial products in the multiplier having said plurality of partial product selectors, the second plurality of partial products comprising four distinct sets of partial products including a first, a second, a third and a fourth set of partial products corresponding to a first, a second, a third and a fourth product for elements of the second set of packed data, and to add the first and second sets of partial products together to produce a first distinct element of a packed result and to add the third and fourth sets of partial products together to produce a second distinct element of the packed result.

Claim 26 sets forth a processor to execute instructions of the PENTIUM microprocessor instruction set<sup>1</sup>, the processor comprising: decode logic<sup>2</sup> to decode a packed sum of absolute differences (PSAD) instruction<sup>3</sup> having a first format to identify a first set of packed data<sup>4</sup>, said decode logic to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction<sup>2</sup>; execution logic to perform a

<sup>1</sup> "In one embodiment of the invention, the processor 105 supports the Pentium® microprocessor instruction set and the packed data instruction set 145. By including the packed data instruction set 145 into a standard microprocessor instruction set, such as the Pentium® microprocessor instruction set, packed data instructions can be easily incorporated into existing software (previously written for the standard microprocessor instruction set)." (Fig. 1, p. 9, line 20 through p. 10, line 1). Pentium® is a registered trademark of Intel Corporation. (p. 10, lines 6-7).

<sup>2</sup> "The decode unit 140 is used for decoding instructions received by the processor 105 into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, the execution unit 142 performs the appropriate operations. The decode unit 140 may be implemented using any number of different mechanisms (e.g., a look-up table, a hardware implementation, a PLA, etc.)." (Fig. 1, p. 9, lines 8-13)

<sup>3</sup> "The decode unit 140 is shown including a packed data instruction set 145 for performing operations on packed data. In one embodiment, the packed data instruction set 145 includes a PMAD instruction(s) 150, a PADD instruction(s) 151, a packed subtract instruction(s) (PSUB) 152, a packed subtract with saturate instruction(s) (PSUBS) 153, a packed maximum instruction(s) (PMAX) 154, a packed minimum instruction(s) (PMIN) 155 and a packed sum of absolute differences instruction(s) (PSAD) 160." (Fig. 1, p. 9, lines 14-20)

<sup>4</sup> "In one embodiment of the invention, the execution unit 142 operates on data in several different packed (non-scalar) data formats. For example, in one embodiment, the exemplary computer system 100 manipulates 64-bit data groups and the packed data can be in one of three formats: a "packed byte" format, a "packed word" format, or a "packed double-word" (dword) format. Packed data in a packed byte format includes eight separate 8-bit data elements. Packed data in a packed word format includes four separate 16-bit data elements and packed data in a packed dword format includes two separate 32-bit data elements." (p. 10, lines 11-18)

first operation of the first set of operations initiated by the decode logic<sup>2</sup>; and a bus<sup>5</sup> to provide the first set of packed data to the execution logic for performing of the first operation.

Claim 39 sets forth a processor comprising: decode logic<sup>2</sup> to decode a packed sum of absolute differences (PSAD) instruction<sup>3</sup> having a first format to identify a first set of packed data<sup>4</sup>, said decode logic to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction<sup>2</sup>, the first set of operations comprising: a packed subtract and write carry (PSUBWC) operation<sup>6</sup>; a packed absolute value and read carry (PABSRC) operation<sup>7</sup>; and a packed add horizontal (PADDH) operation<sup>8</sup>; and execution logic to perform the first set of operations initiated by the decode logic<sup>2</sup>.

Claim 16 sets forth a processor comprising: a decode unit<sup>2</sup> to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction<sup>3</sup> having a first format to identify a first set of packed data<sup>4</sup>, and a packed multiply-add (PMAD) instruction<sup>3,9</sup> having a second format to identify a second set of

<sup>5</sup> "FIG. 1 illustrates that the processor 105 includes a decode unit 140, a set of registers 141, an execution unit 142, and an internal bus 143 for executing instructions. Of course, the processor 105 contains additional circuitry, which is not necessary to understanding the invention. The decode unit 140, the set of registers 141 and the execution unit 142 are coupled together by the internal bus 143." (Fig. 1, p. 9, lines 4-8)

<sup>6</sup> "In step 500, the first operation is a packed subtract and write carry (PSUBWC) operation. For example, in a PSUBWC  $F \leftarrow D, E$  operation, each packed data element  $F_i$  of the packed byte data  $F$  is computed by subtracting the packed data element  $E_i$  of the packed byte data  $E$  from the corresponding packed data element  $D_i$  of the packed byte data  $D$ . Each packed data element in the packed byte data  $D, E$ , and  $F$  represent an unsigned integer. Each carry bit  $C_i$  of a set of carry bits  $C$  is stored. Each carry bit  $C_i$  indicates the sign of the corresponding packed data element  $F_i$ ." (Fig. 5, p. 13, lines 5-11)

<sup>7</sup> "In step 510, the second operation is a packed absolute value and read carry (PABSRC) operation. For example, in a PABSRC  $G \leftarrow 0, F$  operation, each packed data element  $G_i$  of a packed byte data  $G$  is computed by adding a packed data element  $F_i$  of the packed byte data  $F$  to a zero 501 (if the carry bit  $C_i$  indicates the corresponding packed data element  $F_i$  is non-negative) and subtracting the packed data element  $F_i$  from the zero 501 (if the carry bit  $C_i$  indicates the corresponding packed data element  $F_i$  is negative)." (Fig. 5, p. 13, lines 12-18)

<sup>8</sup> "In step 520, the third operation is a packed add horizontal (PADDH) operation. For example, in a PADDH  $R \leftarrow G, 0$  operation, a PMAD circuit is used to produce the result  $RS$  having a field that represents the sum of all of the packed data elements of packed byte data  $G$  as described with reference to FIGS. 11, 12 and 13 below. The PADDH operation is also referred to as a horizontal addition operation." (Fig. 5, p. 13, line 21 through p. 14, line 2)

<sup>9</sup> "FIG. 2 illustrates one embodiment of the PMAD instruction 150. Each packed data element  $A_i$  of a packed word data  $A$  is multiplied by the corresponding packed data element  $B_i$  of a packed word data  $B$  to

packed data<sup>4</sup>, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction<sup>2</sup>; and an execution unit<sup>2,10</sup> to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit<sup>2</sup>.

Claim 17 sets forth the processor of Claim 16, wherein the decode unit further decodes a plurality of instructions of a PENTIUM microprocessor instruction set<sup>1</sup>.

Claim 21 sets forth the processor of Claim 16, wherein performing the first operation causes the execution unit to: produce a first plurality of partial products in a multiplier<sup>11</sup> having a plurality of partial product selectors<sup>12</sup>; insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products<sup>13</sup> by using partial product selectors corresponding to the bit positions<sup>14</sup>; and add the first plurality of elements together to

---

produce doubleword products that are summed by pairs to generate the two packed data elements  $T_0$  and  $T_1$  of a packed dword data T. Thus,  $T_0$  is  $A_1B_1 + A_2B_2$  and  $T_1$  is  $A_3B_3 + A_4B_4$ . As illustrated, the packed data elements of packed dword data T are twice as wide as the packed data elements of the packed word data A and the packed word data B." (Fig. 2, p. 11, lines 9-15)

<sup>10</sup> "FIG. 11 illustrates one embodiment of a PADDH apparatus of the present invention. ... When the CNTR2 signal is deasserted, a PADDH apparatus 1150 performs the PMAD instruction 150." (see Figs. 11 & 12, p. 20, line 9 through p. 26, line 22)

<sup>11</sup> "The portions of the eight selected partial products of the first sixteen partial products and all the bit positions of the remaining partial products on the bus 1101 and the bus 1102 are generated (using prior art partial product selectors or PADDH partial product selectors, for example) as described in the case of the CNTR2 signal being deasserted." (Fig. 11, p. 22, lines 20-24)

<sup>12</sup> "In one embodiment, the set of 16x16 multipliers 1100 use multiple partial product selectors employing Booth encoding to generate partial products. Each partial product selector receives a portion of the multiplicand and a portion of the multiplier and generates a portion of a partial product according to well-known methods." (Fig. 11, p. 21, lines 9-12) "FIG. 13 illustrates one embodiment of a PADDH partial product selector of the present invention." (see Fig. 13, p. 26, line 22 through p. 28, line 2)

<sup>13</sup> "When the CNTR2 signal is asserted, certain partial product selectors (PADDH partial product selectors) within the set of 16x16 multipliers 1100 are configured to insert each packed data element  $G_i$  into a portion of one of the first sixteen partial products." (Fig. 11, p. 22, lines 9-11)

<sup>14</sup> "The PADDH partial product selectors are configured to insert the packed data element  $G_0$  at A10-A17, the packed data element  $G_1$  at B08-B15, the packed data element  $G_2$  at C06-C13, the packed data element  $G_3$  at D04-D11, the packed data element  $G_4$  at I10-I17, the packed data element  $G_5$  at J08-J15, the packed data element  $G_6$  at K06-K13, and the packed data element  $G_7$  at L04-L11." (Fig. 12, p. 24, lines 21-25)

produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit<sup>15</sup>.

Claim 23 sets forth a processor comprising: a decode unit<sup>2</sup> to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction<sup>3</sup> having a first format to identify a first set of packed data<sup>4</sup>, and a packed multiply-add (PMAD) instruction<sup>3,9</sup> having a second format to identify a second set of packed data<sup>4</sup>, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction<sup>2</sup>; and an execution unit<sup>2,10</sup> to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit<sup>2</sup>; wherein performing the first operation causes the execution unit to: produce a first plurality of partial products in a multiplier<sup>11</sup> having a plurality of partial product selectors<sup>12</sup>, insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products<sup>13</sup> by using partial product selectors corresponding to the bit positions<sup>14</sup>, and add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit<sup>15</sup>; and wherein performing the second operation causes the execution unit to: produce a second plurality of partial products in the multiplier<sup>16</sup> having the plurality of partial product

<sup>15</sup> "The CSA tree with CLA 1110 is coupled to receive the first sixteen partial products on the bus 1101 and generate the sum of the first sixteen partial products on the bus 1103. The sum of the first sixteen partial products on the bus 1103 includes the sum all of the packed data elements of the packed data G in a field within the result (see FIG. 12). ... The shifter 1130 performs a right shift operation on the result RS to produce the result R having the field representing the sum all of the packed data elements of packed byte data G aligned with the least significant bit of the result R. In one embodiment, a right shift of RS by 10 bits is used to generate the result R." (see Figs. 11, p. 23, lines 5-19)

<sup>16</sup> The set of 16x16 multipliers 1100 multiply each packed data element Ai of the packed word data A received on the bus 1140 with the corresponding packed data element Bi of the packed word data B received on the bus 1141 to produce thirty-two 18-bit partial products using radix 4 multiplication." (Fig. 11, p. 20, line 13 through p. 21, line 3)

selectors<sup>12</sup>, the second plurality of partial products comprising four distinct sets of partial products including a first set of partial products corresponding to a first product for elements of the second set of packed data, a second set of partial products corresponding to a second product for elements of the second set of packed data, a third set of partial products corresponding to a third product for elements of the second set of packed data, and a fourth set of partial products corresponding to a fourth product for elements of the second set of packed data<sup>17</sup>, and add the first set of partial products together with the second set of partial products to produce a first distinct element of a packed result and add the third set of partial products together with the fourth set of partial products to produce a second distinct element<sup>18</sup> of the packed result.

---

<sup>17</sup> "The eight partial products corresponding to the product of  $A_0$  and  $B_0$  and the eight partial products corresponding to the product of  $A_1$  and  $B_1$  (the first sixteen partial products) are produced on a bus 1101. The eight partial products corresponding to the product of  $A_2$  and  $B_2$  and the eight partial products corresponding to the product of  $A_3$  and  $B_3$  (the second sixteen partial products) are produced on a bus 1102." (Fig. 11, p. 21, lines 3-8)

<sup>18</sup> "A carry-save adder (CSA) tree with carry lookahead adder (CLA) 1110 is coupled to receive the first sixteen partial products on the bus 1101 and generate the sum of the first sixteen partial products on a bus 1103. The sum of the first sixteen partial products on the bus 1103 is the sum of the product of  $A_0$  and  $B_0$  and the product of  $A_1$  and  $B_1$ . The CSA tree with CLA 1120 is coupled to receive the second sixteen partial products on the bus 1102 and generate the sum of the second sixteen partial products on a bus 1104. The sum of the second sixteen partial products on the bus 1104 is the sum of the product of  $A_2$  and  $B_2$  and the product of  $A_3$  and  $B_3$ ." (Fig. 11, p. 21, lines 13-20)

VI. Grounds of Rejection to be Reviewed on Appeal

A. Claims 17 and 26-38 stand rejected under 35 USC § 112, as allegedly being indefinite.

B. Claims 16-20, 25, 26-38 and 39-42 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (Sidwell) in view of in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun).

C. Claims 21-24, 33-34 and 43-44 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (Sidwell) in view of in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun) and further in view of US Patent 5,721,697 (Lee).

VII. Argument

A. 35 U.S.C. § 112 REJECTIONS

Claims 17 and 26-38 stand rejected under 35 USC § 112, second paragraph, as allegedly being indefinite, the Office Action (8.2) stating that through use of the trademark, PENTIUM®, the claim fails to identify any particular material or product, and as a result, renders the claim indefinite.

1. Claims 17 and 26-38 Are Not Indefinite.

With regard to Claims 17 and 26, the Office Action mailed January 10, 2005, states that appellant is "overlooking the clear language of MPEP 2173.05(u)." Appellant

respectfully points out that MPEP 2173.05(u) states that (emphasis supplied):

"The presence of a trademark or trade name in a claim is not, per se, improper under 35 USC § 112, second paragraph, but the claim should be carefully analyzed to determine how the mark or name is used in the claim. It is important to recognize that a trademark or trade name is used to identify a source of goods, and not the goods themselves. ... If the trademark or trade name is used in a claim to identify or describe a particular material or product, the claim does not comply with the requirements of 35 U.S.C. 112, second paragraph. ... Does its presence in the claim cause confusion as to the scope of the claim?"

The Office Action (8.3) incorrectly characterizes a declaration submitted October 11, 2004, suggesting appellant attested that the trademark, PENTIUM, specifically identifies a particular product. Appellant respectfully rebuts the suggestion. Appellant states instead that the phrase, "instructions of the PENTIUM microprocessor instruction set," had, and has, a fixed and definite meaning, and would apprise one skilled in the art of claims 17 and 26's respective scope.

Appellant submits that, as the Office Action (8.2) correctly states, the trademark, PENTIUM, is not being used as descriptive of a particular material or product, which would not be in compliance with 35 USC § 112, second paragraph. Rather, what is set forth is, "instructions of a PENTIUM microprocessor instruction set," which is descriptive of the source of a publicly disclosed microprocessor instruction set. Appellant submits that the microprocessor instruction set associated with that particular source is a well established *de facto* standard of compatibility.

Evidenced by arguments found in the Office Action (8.2), the Examiner apparently understands from MPEP 2173.05(u) that even if the trademark, PENTIUM, is descriptive of the source of a microprocessor instruction set, and is not being used as descriptive of a particular material or product, the use of the trademark, PENTIUM, in claims 17 and 26 would still, necessarily, be improper. Appellant respectfully disagrees.

Appellant submits that MPEP 2173.05(u) cites a decision of the Board of Patent

42390P5943C

-12-



Appeals and Interferences in *Ex parte Simpson*, 218 U.S.P.Q. 1020 (Bd. App. 1982), where the term "Hypalon" was being used (improperly) as a noun in a claim to describe the physical and/or other properties of a material. The Board ruled that, "[t]he claim scope [was] uncertain as regards the material which forms the 'Hypalon.'"

Such is not the case with the present claims 17 and 26 where the trademark, "PENTIUM," is being used as an adjective descriptive of the source of a well known, publicly disclosed, microprocessor instruction set.

Appellant respectfully argues that the presence of a trademark or trade name in a claim is not improper under 35 USC § 112, second paragraph, if its presence in the claim does not cause confusion as to the scope of the claim.

Claim 17, for example, sets forth:

17. (Previously Presented) The processor of Claim 16, wherein the decode unit further decodes a plurality of instructions of a PENTIUM microprocessor instruction set.

Claim 26, for example, also sets forth:

26. (Previously Presented) A processor to execute instructions of the PENTIUM microprocessor instruction set, the processor comprising:  
    decode logic to decode a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, said decode logic to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction;  
    execution logic to perform a first operation of the first set of operations initiated by the decode logic; and  
    a bus to provide the first set of packed data to the execution logic for performing of the first operation.

Claims 17 and 26 set forth, respectively, a decode unit to decode and a processor to execute instructions of the PENTIUM microprocessor instruction set. Appellant respectfully submits that the PENTIUM microprocessor instruction set identifies the source of the instruction set in such a way that the scope of the subject matter embraced by the claim is fixed and definite. Further, the PENTIUM microprocessor instruction set

was known generally to those skilled in the art at the time the original application was filed. For example, numerous software engineers and hardware engineers relied and still rely upon the publicly available definition of the PENTIUM microprocessor instruction set in order to conduct business and to plan engineering projects.

Appellant respectfully submits as further evidence of the above conclusion the accompanying exhibits (emphasis supplied):

- i. The "Pentium® Processor Family Developer's Manual, Vol. 3: Architecture and Programming Manual," 1995, pp. 25-165 and 25-166, cited by the Examiner in the Office Action (8.5) as extrinsic evidence for a common knowledge of the PENTIUM microprocessor instruction set by one of ordinary skill in the art.
- ii. A November 1997 article by Eric Traut from BYTE, which discusses a Macintosh application that employs a "Pentium instruction-set emulator, complete with MMX<sup>TM</sup> instructions."
- iii. A definition of AMD from wordIQ.com, which explains (in the History section, paragraph 7) that at some time about one year after AMD purchased NexGen in 1996, "the K6 [processor] translated the Pentium compatible x86 instruction set to RISC-like micro-instructions."
- iv. John Savill's FAQ (Frequently Asked Questions) for Windows web page, dated September 3, 1999, which asks, "Do I really need 166Mhz Pentium processors to run SQL Server 7.0?" The answer given states, "No. But you DO need a 100% PENTIUM compatible chip - which rules out some Cyrix and IBM processors." The page further explains (in paragraph 3) that, "speed of the processor doesn't matter as long as it runs the full pentium instruction set."
- v. A current product description of a single-board computer from SBS technologies, which includes a "Pentium compatible Geode GX1 processor."
- vi. A Department of Energy (hq.doe.gov) description of the Hardware & System Requirements for Microsoft Windows 2000 and Microsoft Office 2000 by IT Standards Manager, Carol Blackston, requiring a "133 MHz or higher Pentium-compatible CPU" for Windows 2000 Professional, a 166 MHz Pentium-compatible CPU or higher" for Office 2000 Premium, and a "75 MHz Pentium-compatible CPU or higher" for Office 2000 Professional or Office 2000 Standard.
- vii. Microsoft requirements for a Microsoft Operations Manager Server, a Database Server, a Reporting Server, or a SQL Server 2000 Reporting Services Server, listed as a "PC with 550 MHz or higher Pentium-compatible;" an Administrator and Operator Console, listed as a "PC with 500 MHz or higher Pentium-compatible;" and a Managed Computer, listed as a "PC with 200 MHz or higher Pentium-compatible."
- viii. An article by Taran Rampersad from the Free Software Consortium (FSC) dated March 26, 2004, describing the basic system requirements of OpenOffice under Windows (98, NT, 2000, XP) including a "Pentium-compatible PC."
- ix. An article by Thomas Latuske posted June 8, 2004, describing two ways to retrieve the processor-speed and stating (in paragraph 1) that, "If you want to use the function to calculate the speed (frequency), you have to use it with a Pentium instruction set compatible processor."

According to the above references (i, ii and iii) prior art disclosed the PENTIUM microprocessor instruction set, the instruction set was known to persons skilled in the art, and it was readily obtainable at the time the application was filed. As such, use of the

trademark is not improper under 35 USC § 112. *Leutzing v. Ladd*, 222 F. Supp. 681, 682. Appellant concludes from the above references (i through ix) that as of the filing date in March of 1998, the phrase, "instructions of the PENTIUM microprocessor instruction set," had (and continues to have today) a fixed and definite meaning, and therefore, would apprise one skilled in the art of the respective scope of claim 17 and of claim 26.

Appellant notes that in MPEP 608.01(v), par. 6, it states (emphasis supplied):

"If the product to which the trademark refers is set forth in such language that its identity is clear, the examiners are authorized to permit the use of the trademark if it is distinguished from common descriptive nouns by capitalization. If the trademark has a fixed and definite meaning, it constitutes sufficient identification unless some physical or chemical characteristic of the article or material is involved in the invention."

Therefore, when the trademark has a fixed and definite meaning, it constitutes sufficient identification in accordance with MPEP 608.01(v).

The amount of detail required to be included in claims depends on the particular invention and the prior art, and is not to be viewed in the abstract but in conjunction with whether the specification is in compliance with the first paragraph of section 112.

*Chemcast Corp. v. Arco Industries Corp.*, 854 F.2d 1328 (Fed. Cir. 1988).

The present specification discloses (p. 9, line 20 through p. 10, line 1) that:

"In one embodiment of the invention, the processor 105 supports the Pentium® microprocessor instruction set and the packed data instruction set 145. By including the packed data instruction set 145 into a standard microprocessor instruction set, such as the Pentium® microprocessor instruction set, packed data instructions can be easily incorporated into existing software (previously written for the standard microprocessor instruction set)."

The Court of Customs & Patent Appeals, considering the use of a trade name, "Pliolite," in a claim in conjunction with the first paragraph requirements of section 112, where (appellant respectfully takes note that) neither the composition of "Pliolite" or "Plioform," nor a method of preparing them, nor who manufactured or sold them, was

disclosed in the original application, held that, "A fair interpretation of the facts herein leads to the conclusion that the Goodyear products 'Plioform' and 'Pliolite' were on the market and were known generally to those skilled in the art of chemistry at the time the original application was filed by appellants. With the information contained in the original application, therefore, it was possible for those skilled in the art at that time to practice appellants' invention, and thus it follows that the disclosure therein was sufficient." *In re Gebauer-Fuelnegg*, 50 USPQ 125 (C.C.P.A. 1941).

Therefore, appellant respectfully submits that the specification has set forth a full and clear description of the claimed subject matter in sufficient detail to conclude that appellant had possession of the claimed invention, further to enable one skilled in the art to practice the claimed invention, and finally to apprise one skilled in the art of claims' respective scope.

In addition, the Office Action (8.2) maintains that there are at least ten (10) different particular microprocessors produced by Intel Corporation which carry the trademark, PENTIUM, many of which contain different instruction sets. While appellant disagrees that these particular microprocessors contain wholly different instruction sets, appellant respectfully submits that a claim is not indefinite simply because it covers a number of possible embodiments.

The Court of Customs & Patent Appeals in considering the expression "organic and inorganic acids," which was alleged to be indefinite and of uncertain scope, held that, "Although there are undoubtedly a large number of acids which come within the scope of 'organic and inorganic acids,' the expression is not for that reason indefinite." *In re Skoll*, 187 USPQ 481 (CCPA 1975).

Appellant therefore submits that Claims 17 and 26 set out and circumscribe subject matter with a sufficient degree of precision and particularity to apprise one of skill in the art of each claim's respective scope.

Accordingly in light of the argument presented above, appellant respectfully submits that claims 17 and 26-38 are not indefinite and are, therefore, in compliance with 35 USC § 112, second paragraph.

**B. FIRST 35 U.S.C. § 103(a) REJECTIONS**

Claims 16-20, 25, 26-38 and 39-42 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (hereafter "Sidwell") in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (hereafter "Sun").

**1. Claims 17 and 26-29 Are Not Obvious.**

With regard to Claims 17 and 26-29, the Office Action of Aug. 19, 2003 (8 of paper no. 5) states that it would have been obvious to make a combined system of Sidwell and Sun, perform Sun's packed sum of absolute differences, compatible with the PENTIUM® microprocessor instruction set. Appellant respectfully disagrees.

First, in determining the scope and content of the cited references with regard to the instant claims at issue, appellant respectfully submits that Sidwell is directed to an arithmetic unit for packed arithmetic. The arithmetic unit of Sidwell is comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19). Two source operands for the packed arithmetic units 70-80 are supplied along the Source 1 and Source 2 busses 52, 54 (col. 5, lines 26-27). Sidwell discloses that one separate packed arithmetic execution unit (the multiply-add unit 76) is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied). Sidwell discloses that another separate packed arithmetic execution

42390P5943C

-18-

unit (the obvious packed arithmetic unit 80) performs the addition, subtraction, comparison and multiplication of packed numbers (Figs. 4 and 5, col. 5, line 50 through col. 6, line 47).

Sun is directed to a set of visual instructions that are used primarily to write graphics and multimedia applications (p. 41, first paragraph). One of these instructions (the vis\_pdist() instruction) accumulates the absolute values of differences into a destination accumulator (p. 87, last paragraph). It is also shown by Sun that there is not a sum of absolute values without accumulation and therefore the accumulator must be initialized to zero prior to beginning execution of the vis\_pdist() instructions (p. 88, line 9, 4.7.11 Example). The vis\_pdist() instruction of Sun has three source operands, one of which is also a destination and it is necessary for the accumulating register, *accumulator*, to appear both as an argument and as the receiver of the return value (p. 88, first paragraph).

Next, appellant respectfully points out some of the differences between the cited references and the instant claims at issue. Claim 17, for example, sets forth:

17. (Previously Presented) The processor of Claim 16, wherein the decode unit further decodes a plurality of instructions of a PENTIUM microprocessor instruction set.

And Claim 16 sets forth:

16. (Previously Presented) A processor comprising:  
a decode unit to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, and a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction; and  
an execution unit to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit.

In the Office Action of Aug. 19, 2003 (7 and 8 of paper no. 5) the Examiner asserts that it would have been obvious to combine Sun's packed sum of absolute differences to Sidwell's system because Sidwell taught that the packed arithmetic unit performed additional operations (col. 5, lines 15-22) and Sun taught that a packed sum of absolute differences instruction was beneficial in accelerating motion compensation to support real-time video compression (p. 88) and then to make the combined system of Sidwell and Sun, perform Sun's vis\_pdist() instruction, compatible with the PENTIUM® microprocessor instruction set because the PENTIUM microprocessor instruction set is the most widely used microprocessor instruction set in the world.

The multiply-add instruction of Sidwell (muladd2ps) employs three operands (e.g. see Sidwell, col. 8, lines 24 and 34-37). The vis\_pdist() instruction of Sun also requires three operands one of which is both a source and a destination (e.g. see Sun, p. 88, line 10). Therefore, one difference between the claimed decoder of instructions of the PENTIUM microprocessor instruction set and the expected properties of the combined system of Sidwell and Sun is the absence of an expected third operand.

Appellant respectfully submits that since the PENTIUM microprocessor instruction set has a well known opcode format, which permits two operands, one of the operands acting both as a source operand and a destination operand. To decode and execute the vis\_pdist() instruction of Sun having three source operands in a processor for executing two-operand instructions of the PENTIUM microprocessor instruction set is not suggested by either of the cited references.

Further, since the vis\_pdist() instruction of Sun explicitly requires at least three



sources (two sets of pixels and one accumulator) the use of a two-operand format would be unexpected. Yet, because the instructions of the PENTIUM microprocessor instruction set require only two sources, one of which is also a destination, the data path for the packed sum of absolute differences may be 75% as wide as one requiring three sources and the number of read ports required in the register file may be 66% as many as would otherwise be required for reading three source registers. Such reductions are statistically significant. Moreover, in order for a decoder of the PENTIUM microprocessor instruction set to be adapted to also decode a new three-operand instruction, significant modifications and increased design complexity would be required, most probably introducing additional delays to critical timing paths for high frequency designs. Such considerations are also of great practical significance in the field of microprocessor design.

The Final Office Action of April 9, 2004 (7.3 of paper no. 8) states that there are many PENTIUM instructions that while having only two programmer specified operands, make use of one or more additional implicit operands, and so, as a result that instruction is effectively a three or more operand instruction, citing as an example the IMUL instruction.

The Office Action of Jan. 10, 2005 (8.5) maintains the assertion and provides, as extrinsic evidence for common knowledge in the art of the PENTIUM microprocessor instruction set, a reference, "Pentium® Processor Family Developer's Manual, Vol. 3: Architecture and Programming Manual," 1995, pp. 25-165 and 25-166, showing IMUL instructions that uses implicit operands.

Appellant respectfully notes that the three IMUL instructions listed that use

implicit operands use only one programmer specified operand and an implicit destination register where the lower half of the destination register is used as a source (p. 25-165, lines 3-5). Thus, the IMUL instructions with implicit operands are effectively, still, two-operand instructions.

Therefore, appellant respectfully submits that since the PENTIUM microprocessor instruction set has an opcode format that permits two operands, one of those operands acting both as a source operand and a destination operand; and since the vis\_pdist() instruction of Sun requires three source operands, one of which is also a destination (p. 87, 4.7.11 Syntax and Description); a sum of absolute differences instruction with a two-operand opcode format would not perform the operation defined by the vis\_pdist() instruction of Sun without modification. No suggestion of such modification is provided either by the cited references or by a common knowledge of IMUL in one of ordinary skill in the art.

For example, Sun discloses that in the vis\_pdist() instruction, one source is also an accumulating destination register. Therefore, the vis\_pdist() instruction of Sun computes an accumulation of current and prior absolute differences rather than a sum of the absolute differences on a first identified set of packed data as set forth in the instant claims at issue. There is no accumulation of prior absolute differences in what appellant has done. Thus, appellant submits that an absence in the claimed invention of the expected accumulation of prior absolute differences from the combined system of Sidwell and Sun is evidence of nonobviousness.

Additionally, Sun discloses that in the vis\_pdist() instruction, "it is necessary for the accumulating register to appear both as an argument and as the receiver of the return

value" (p. 88, first paragraph, emphasis supplied). Thus, Sun teaches away from an implicit source that is also the destination register, which is precisely the technique employed in the implicit-operand IMUL instructions, of the PENTIUM microprocessor instruction set. Therefore, it would not be obvious to combine the vis\_pdist() instruction of Sun with the implicit operand technique used in the IMUL instructions, when Sun, itself, teaches away from such a technique.

Further, while the Office Action (8.5) maintains that it would have been obvious to use implicit operands, as in the IMUL instruction of the PENTIUM microprocessor instruction set, for the combined system of Sidwell and Sun to perform Sun's packed sum of absolute differences; appellant respectfully submits that the packed sum of absolute differences (PSAD) instruction of the present application does not have an implicit operand. Thus, appellant submits that the absence of an expected implicit operand is also evidence of nonobviousness.

The claims set forth a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, initiating a first set of operations on the first set of packed data responsive to decoding the PSAD instruction, and execution logic to perform a first operation of the first set of operations.

Since Sun teaches away from an implicit source that is also the destination register, the alleged two-operand combination of Sidwell and Sun would necessarily fail to identify some of the first set of packed data, and instead that source of packed data would necessarily have to be the implicit operand.

The packed sum of absolute differences instruction of the present application has a format to identify the first set of packed data on which to perform the packed sum of

absolute differences. There is no implicit source operand in what appellant has done.

Appellant respectfully submits that the alleged combination of references should not be considered obvious if it does not fairly disclose doing what appellant has done.

The general rule applicable to a rejection based on a combination of references was stated in *In re Schaffer*, 108 USPQ 326, 328-29 (1956):

[References] may be combined for the purpose of showing that a claim is unpatentable. However, they may not be combined indiscriminately, and to determine whether the combination of references is proper, the following criterion is often used: namely, whether the prior art suggests doing what an applicant has done. ... [It] is not enough for a valid rejection to view the prior art in retrospect once an applicant's disclosure is known. The art applied should be viewed by itself to see if it fairly disclosed doing what an applicant has done.

Finally, Sidwell's system provides no path for an accumulator input to packed arithmetic unit 6, either from result bus 56 or as a third source operand to packed arithmetic unit 6 (Figs. 1, 2, 4, and 6; col. 5, line 15 through col. 7, line 53). Therefore, Sidwell's system could not perform Sun's packed sum of absolute differences without significant modifications to permit a third source operand for packed arithmetic instructions. Appellant respectfully submits that no suggestion for such modifications is provided by Sidwell; and even if the modifications were made to Sidwell to permit a third source operand for packed arithmetic instructions, it would be nonobvious, without the aid of appellant's disclosure to view the prior art in retrospect, to perform an operation requiring such a third source operand but using only a two-operand opcode format as in the PENTIUM microprocessor instruction set.

Therefore, appellant respectfully submits that without viewing the prior art in retrospect with the aid of appellant's disclosure, no suggestion is provided by Sidwell or Sun for doing what appellant has done.

Accordingly in light of the above arguments, Claims 17 and 26-29, are not

obvious in view of the cited references.

2. Claims 18, 30 and 39-42 Are Not Obvious.

With regard to Claims 18, 30 and 39-42, the Office Action mailed Aug. 19, 2003 (9 of paper 5) states that Sun taught performing a packed subtract and write carry, a packed absolute value and a packed add horizontal. Appellant respectfully disagrees and again notes that Claims 18, 30 and 39-42 set forth a packed subtract and write carry operation, a packed absolute value and read carry operation, and a packed add horizontal operation (emphasis added).

Claim 39, for example, sets forth:

39. (Previously Presented) A processor comprising:  
decode logic to decode a packed sum of absolute differences (PSAD)  
instruction having a first format to identify a first set of packed data, said decode logic to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction, the first set of operations comprising:  
a packed subtract and write carry (PSUBWC) operation;  
a packed absolute value and read carry (PABSRC) operation; and  
a packed add horizontal (PADDH) operation.; and  
execution logic to perform the first set of operations initiated by the decode logic.

In determining the scope and content of the cited references with regard to the instant claims at issue, appellant respectfully submits that Sun is directed to a set of visual instructions used primarily to write graphics and multimedia applications (p. 41, first paragraph). One of these instructions (the vis\_pdist() instruction) accumulates the absolute values of differences into a destination accumulator (p. 87, last paragraph). Sun states that "the pixels are subtracted from one another, pair wise, and the absolute values of the differences are accumulated into *accum*" (p.87, 4.7.11, Description, first paragraph). Sun does not teach a packed subtract and write carry operation or a packed absolute value and read carry operation, as set forth in claims 18, 30 and 39.

Sidwell is directed to an arithmetic unit for packed arithmetic. The arithmetic unit 6 of Sidwell is comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19). Sidwell discloses that the multiply-add unit 76 is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied). Sidwell discloses that the obvious packed arithmetic unit 80 performs the addition, subtraction, comparison and multiplication of packed numbers (Figs. 4 and 5, col. 5, line 50 through col. 6, line 47). Sidwell states that "The execution units 2, 4, 6 do not hold any state between instructions. Thus subsequent instructions are independent." (col. 4, lines 36-38)

Therefore, Sidwell teaches away from an execution unit to hold carry state between instructions, which is what appellant has done in the packed subtract and write carry operation, and the packed absolute value and read carry operation as set forth by claims 18, 30 and 39.

Appellant respectfully points out some of the differences between the cited references and the instant claims at issue. Sidwell does not disclose a packed sum of absolute differences instruction. Sun discloses that in the `vis_pdist()` instruction, one source is also an accumulating destination register. Therefore, in the combined system of Sidwell and Sun, the `vis_pdist()` instruction would be expected to compute an accumulation of current and prior absolute differences rather than a sum of the absolute differences on a first identified set of packed data as set forth in the instant claims at issue. There is no accumulation of prior absolute differences in the packed sum of

absolute differences of the instant claims at issue. Thus, appellant submits that an absence in the claimed invention of the expected accumulation of prior absolute differences from the combined system of Sidwell and Sun would be unexpected.

Because the packed sum of absolute differences instruction does not require an accumulator source, the data path for the packed sum of absolute differences may be 75% as wide as one requiring a third source and the number of read ports required in the register file may be 66% as many as would otherwise be required for reading a third source register. Such reductions are statistically significant. Of practical significance is that since there is no version of the `vis_pdist()` instruction that does not use an accumulation of prior absolute differences, an additional instruction, `vis_fzero()`, is required to initialize the accumulator before the `vis_pdist()` instruction can be used (e.g. see Sun, p. 88, line 9-10).

Additionally, neither Sidwell nor Sun disclose a packed subtract and write carry operation or a packed absolute value and read carry operation, as set forth in claims 18, 30 and 39. Neither cited reference discusses or suggests the writing of any carry state as part of a packed subtraction operation or the reading of any carry state as part of a packed absolute value operation. In fact, Sidwell teaches away from an execution unit to hold carry state between instructions. Therefore, the presence of such operations in the combined system of Sidwell and Sun would be unexpected.

The Office Action mailed Jan. 10, 2004 (8.6) states that the fact that Sidwell teaches that the execution units do not hold state is immaterial because no execution unit holds state in any system. What Sidwell teaches is (col. 4, lines 36-38, emphasis supplied) that, "The execution units 2, 4, 6 do not hold any state between instructions."



"Thus subsequent instructions are independent." Therefore, appellant respectfully submits that the presence of such unexpected operations to hold carry state between subsequent instructions is evidence of nonobviousness.

Because the packed subtract and write carry operation and the packed absolute value and read carry operation employ an execution unit to hold carry state between subsequent instructions rather than duplicating the adder/subtractor circuitry, the execution circuitry for performing the packed subtract and write carry operation and the packed absolute value and read carry operation may be 50% of the execution circuitry for performing an absolute difference operation without an execution unit to hold carry state between instructions. Such a reduction is statistically significant. The present specification, for example, discloses (Figs. 9 and 10, p. 19, lines 1-8) that:

"In one embodiment, the PSUBWC/PABSRC arithmetic element 900 is the same circuitry used to perform the PADD instruction 151. The mux 920 is added and the  $C_{output,0}$  bus is routed to the register 940 and the  $C_{input,0}$  bus is routed to the mux 920 to provide for the PSAD instruction 160.

By saving the carry bits from the PSUBWC operation and using the saved carry bits to control the subsequent PABSRC operation, the same circuitry used to perform the PADD hardware may be used to perform both the PSUBWC and the PABSRC operations with relatively little additional circuitry."

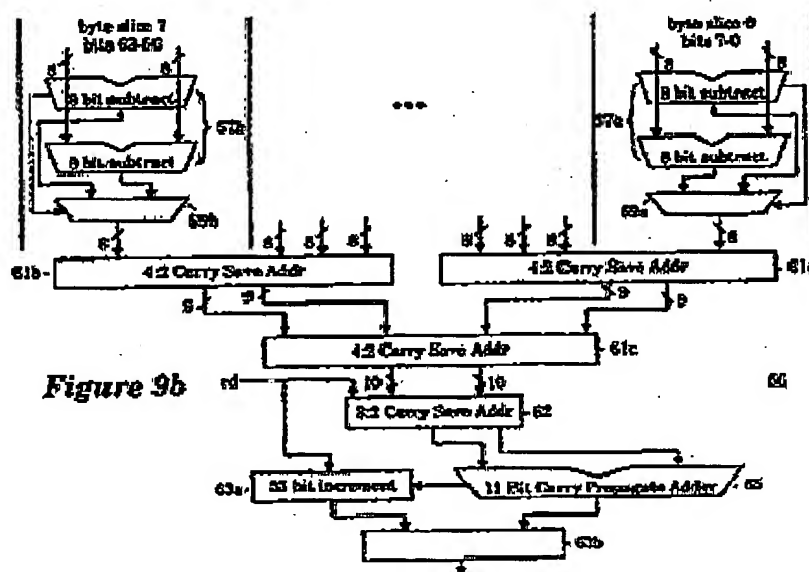
Therefore, through a modification of the PADD hardware to support saving the carry bits from one operation and using the saved carry bits to control a subsequent operation, the same circuitry may be reused and no new circuitry to perform an absolute difference operation is required, which has substantial practical significance.

The Final Office Action of April 9, 2004 (7.4 of paper no. 8) states that a packed subtract and write carry operation and a packed absolute value and read carry operation are inherently present in Sun's system. Appellant respectfully disagrees.

Two commonly used techniques for computing an absolute differences found in

the design of floating point mantissa arithmetic are: (1) compare two numbers and reorder to subtract the smaller from the larger, or (2) subtract the two numbers in both directions and select the positive result. Appellant respectfully submits that one of these two alternatives may reasonably be expected to be inherently present in Sun's system rather than the packed subtract and write carry operation and the packed absolute value and read carry operation set forth by the present application.

For example, appellant respectfully points out that in US Patent 5,938,756 (hereafter "Van Hook") it shows the vis\_pdist() circuit illustrated below:



In the disclosure of Van Hook (Figs. 9a-9b, col. 10 line 53 through col. 11, line 4, emphasis supplied) it also states:

"Referring now to FIGS. 9a-9b, the pixel distance computation instructions, and the pixel distance computation circuit are illustrated. As shown in FIG. 9a, there is one graphics data distance computation instruction 138 for simultaneously accumulating the absolute differences between graphics data, eight pairs at a time. The PDIST instruction 138 subtracts eight 8-bit graphics data in the rs1 register from eight corresponding 8-bit graphics data in the rs2 register. The sum of the absolute values of the differences is added to the content of the rd register. The PDIST instruction is typically used for motion estimation in video compression algorithms.

As shown in FIG. 9b, in this embodiment, the pixel distance computation circuit 36 comprises eight pairs of 8 bit subtractors 57a-57h. Additionally, the pixel distance computation circuit 56 further comprises three 4:2 carry save adders 61a-61c, a 3:2 carry save adder 62, two registers 63a-63b, and a 11-bit carry propagate adder 65, coupled to each other as shown."

Therefore, appellant submits that alternative (2), subtracting the two numbers in both directions and selecting the positive result, is what is likely to be inherent in Sun. As appellant indicates above, that alternative requires new circuitry to perform an absolute difference operation, which has twice as many adder/subtractors (e.g. see Sun, Fig. 9b, 57a-h) as the execution circuitry for performing a packed subtract and write carry operation and a packed absolute value and read carry operation (e.g. see Fig. 10, 1000-1070 of the present application).

Thus, the presence of a packed subtract and write carry operation and a packed absolute value and read carry operation in the combined system of Sidwell and Sun would be nonobvious.

Additionally, the instant claims at issue set forth decode logic to initiate a set of operations responsive to decoding the PSAD instruction, the operations comprising: a packed subtract and write carry (PSUBWC) operation; a packed absolute value and read carry (PABSRC) operation; and a packed add horizontal (PADDH) operation. For example, in the present application (Fig. 1, p. 9, lines 8-10) it states:

"The decode unit 140 is used for decoding instructions received by the processor 105 into control signals and/or microcode entry points."

and further (Fig. 5, p. 13, lines 5-21) states that :

"In step 500, the first operation is a packed subtract and write carry (PSUBWC) operation. ...In step 510, the second operation is a packed absolute value and read carry (PABSRC) operation. ...In step 520, the third operation is a packed add horizontal (PADDH) operation."

Decoding instructions into such sets of control signals and/or microcode entry points is not disclosed by Sidwell or by Sun. Appellant respectfully submits that a

presence in the claimed invention of decode logic to initiate the packed subtract and write carry (PSUBWC), the packed absolute value and read carry (PABSRC), and the packed add horizontal (PADDH) operations responsive to decoding the PSAD instruction, which are not disclosed by the combined references of Sidwell and Sun, would be unexpected without viewing the prior art in retrospect with the aid of appellant's disclosure.

Appellant respectfully submits that Van Hook is relevant to what would be expected from the combined system of Sidwell and Sun to execute the vis\_pdist() instruction. To that end, appellant respectfully points out that in Fig. 9b of Van Hook shown above, a pixel distance computation circuit of the second partitioned execution path is designed to perform the entire vis\_pdist() instruction rather than a microcode operation for a portion of the vis\_pdist() instruction.

Because the PSAD instruction is decoded into a set of microcode control signals, 66% less instructions may need to be fetched and decoded for computing a packed sum of absolute differences, which is statistically significant. Of practical significance, is that not all (if any) of the operations PSUBWC, PABSRC and PADDH, may actually need to be supported outside of the microcode by opcodes for user programmable instructions. Yet, reuse of the PADD hardware for PSUBWC and PABSRC, and of the PMAD hardware for PADDH may be accomplished through use of a microcode sequence. Neither Sidwell nor Sun disclose decoding the vis\_pdist() instruction into a set of microcode operations or the reuse of packed adder hardware or packed multiply-add hardware for computing a packed sum of absolute differences.

Therefore, appellant respectfully submits that without viewing the prior art in retrospect with the aid of appellant's disclosure, no suggestion is provided by Sidwell or

Sun for doing what appellant has done.

Accordingly in light of the above arguments, Claims 18, 30 and 39-42, are not obvious in view of the cited references.

### 3. Claims 16 and 36 Are Not Obvious.

With regard to Claims 16 and 35, the Office Action states that where modifications would be required in the combined references, appellant is in error to suggest that the references must disclose or suggest such modifications, and that the measure is what the teachings would suggest to one of ordinary skill in the art. Appellant respectfully disagrees with the Examiner's characterization of what Sidwell and Sun taught or would suggest to one of ordinary skill in the art.

Once again, in determining the scope and content of the cited references with regard to the instant claims at issue, appellant respectfully submits that Sidwell is directed to an arithmetic unit for packed arithmetic. The arithmetic unit 6 of Sidwell is comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19). Sidwell discloses that the obvious packed arithmetic unit 80 performs the addition, subtraction, comparison and multiplication of packed numbers (Figs. 4 and 5, col. 5, line 50 through col. 6, line 47). Sidwell discloses that the multiply-add unit 76 is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied). Sidwell does not disclose or suggest reuse of the multiply-add unit 76 even for performing packed multiply instructions (mul2ps). Nor does Sidwell disclose or suggest cooperation between the various collection of separate packed arithmetic units to perform any of the packed instructions.

Sun is directed to a set of visual instructions used primarily to write graphics and

multimedia applications (p. 41, first paragraph). One of these instructions (the vis\_pdist() instruction) accumulates the absolute values of differences into a destination accumulator (p. 87, last paragraph). Sun states that "the pixels are subtracted from one another, pair wise, and the absolute values of the differences are accumulated into *accum*" (p.87, 4.7.11, Description, first paragraph). Sun does not disclose or suggest a version of the vis\_pdist() instruction that does not use an accumulation of prior absolute differences.

Appellant now respectfully points out some of the differences between the cited references and the instant claims at issue. Claim 16, for example, sets forth:

16. (Previously Presented) A processor comprising:  
a decode unit to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, and a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction; and  
an execution unit to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit.

Sidwell does not disclose a packed sum of absolute differences instruction. Sun discloses that in the vis\_pdist() instruction, one source is also an accumulating destination register. Therefore, in the combined system of Sidwell and Sun, the vis\_pdist() instruction would be expected to compute an accumulation of current and prior absolute differences rather than a sum of the absolute differences on the first identified set of packed data as set forth in the instant claims at issue. There is no accumulation of prior absolute differences in the packed sum of absolute differences of the instant claims at issue. Thus, appellant submits that an absence in the claimed invention of the expected accumulation of prior absolute differences from the combined system of Sidwell and Sun

would be unexpected.

For example, Sun shows a diagram of a floating-point & graphics unit to perform the vis\_pdist() instruction (Fig. 2-4, p. 11) as follows:

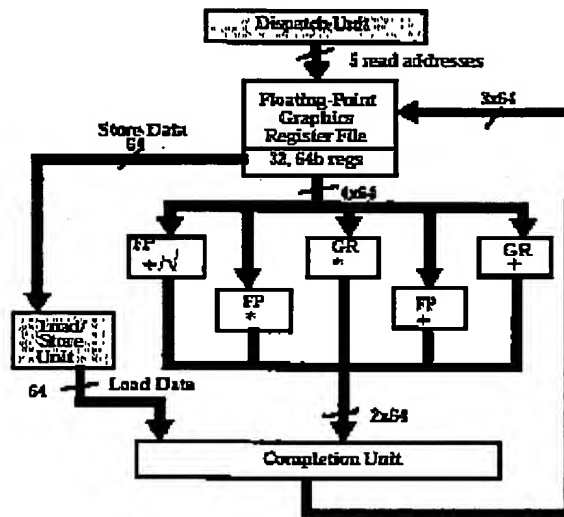
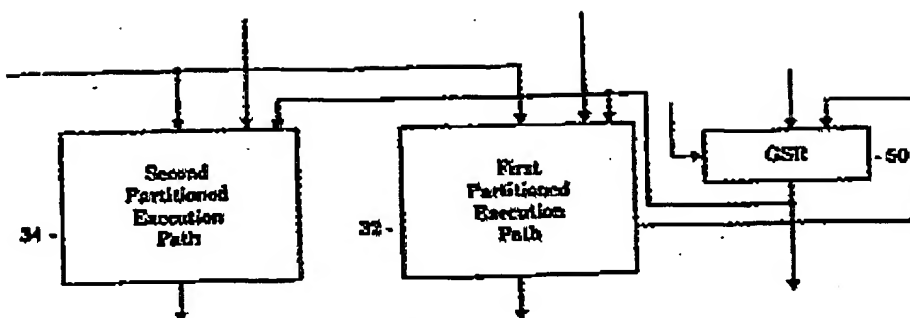


Figure 2-4 Floating Point and Graphics Unit

Sun discloses that the graphics adder (GR+) and graphics multiplier (GR\*) perform the graphics operations of the VIS instruction set (p. 11, lines 10-11).

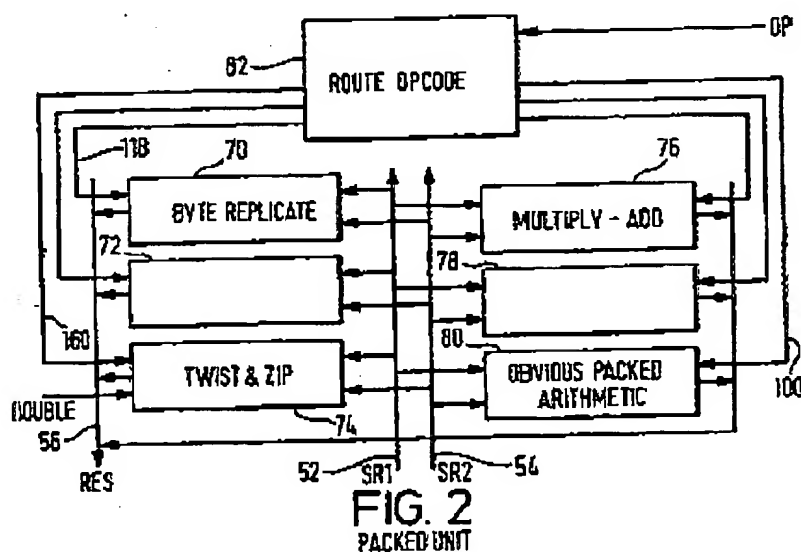
Appellant again refers to the disclosure of US Patent 5,938,756, Van Hook. Like Sun, Van Hook shows a diagram of a graphics execution unit (GRU) having first (corresponding to GR+) and second (corresponding to GR\*) independent execution paths to independently execute graphics instructions (Fig. 2, col. 4, lines 3-10) as follows:



**Figure 2**

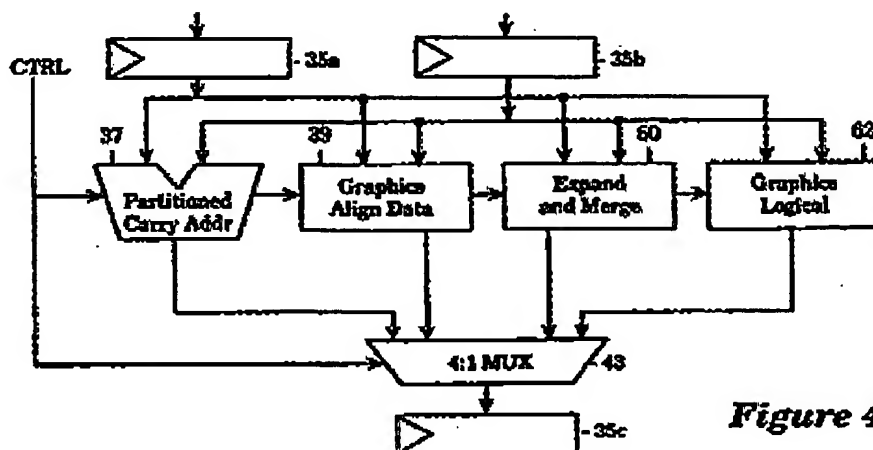
Van Hook discloses that the second partitioned execution path comprises a pixel distance computation circuit (Fig. 5, 36, col. 5, lines 4-5).

Sidwell shows a diagram of an arithmetic unit comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19) as follows:



Like Sidwell, Van Hook shows a diagram of his first partitioned execution path

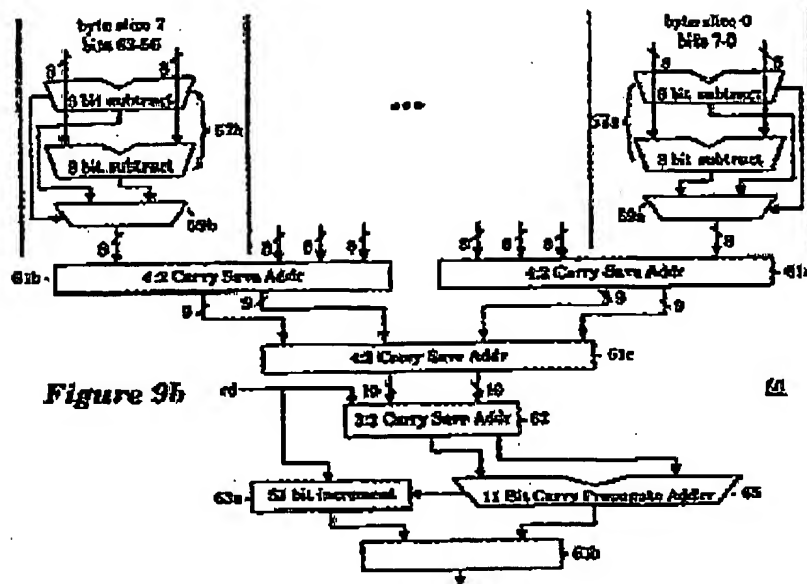
similarly comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 4, col. 4, lines 26-46) as follows:



**Figure 4**

For example, the obvious packed arithmetic unit 80 of Sidwell may perform some subset of packed arithmetic instructions similar to those performed by the partitioned carry adder 37 of Van Hook (e.g. see Van Hook, col. 4, lines 47-54). Therefore, appellant respectfully submits that Van Hook is relevant to what would be expected from the combined system of Sidwell and Sun, with regard to an included `vis_pdist()` instruction.

To that end, appellant respectfully points out that Van Hook shows a diagram of the pixel distance computation circuit of the second partitioned execution path to perform the `vis_pdist()` circuit as illustrated below:



In the disclosure of Van Hook (Figs. 9a-9b, col. 10 line 53 through col. 11, line 4, emphasis supplied) it also states:

"Referring now to FIGS. 9a-9b, the pixel distance computation instructions, and the pixel distance computation circuit are illustrated. As shown in FIG. 9a, there is one graphics data distance computation instruction 138 for simultaneously accumulating the absolute differences between graphics data, eight pairs at a time. The PDIST instruction 138 subtracts eight 8-bit graphics data in the rs1 register from eight corresponding 8-bit graphics data in the rs2 register. The sum of the absolute values of the differences is added to the content of the rd register. The PDIST instruction is typically used for motion estimation in video compression algorithms.

As shown in FIG. 9b, in this embodiment, the pixel distance computation circuit 36 comprises eight pairs of 8 bit subtractors 57a-57h. Additionally, the pixel distance computation circuit 56 further comprises three 4:2 carry save adders 61a-61c, a 3:2 carry save adder 62, two registers 63a-63b, and a 11-bit carry propagate adder 65, coupled to each other as shown."

Therefore, appellant submits that an absence in the claimed invention of the expected accumulation of prior absolute differences from the combined system of Sidwell and Sun would be unexpected.

Because the packed sum of absolute differences instruction does not require an accumulator source, the data path for the packed sum of absolute differences (requiring

only two sources and one destination) may be 75% as wide as one also requiring a third source, which is statistically significant. Of practical significance is that the number of computational stages required would be five for computing just a sum of absolute differences (e.g. eliminating 3:2 carry save adder 62 of Van Hook) instead of six as shown by Van Hook, which could impact the maximum design frequency. Also since there is no version of the vis\_pdist() instruction that does not use an accumulation of prior absolute differences, an additional instruction, vis\_fzero(), is required by Sun to initialize the accumulator before the vis\_pdist() instruction can be used (e.g. see Sun, p. 88, line 9-10).

Thus, the absence of the expected accumulation of prior absolute differences from the combined system of Sidwell and Sun would be nonobvious.

Claim 6 also sets forth an execution unit to perform a first operation of the first set of operations initiated by the decode unit responsive to decoding the PSAD instruction and to perform a second operation of the second set of operations initiated by the decode unit responsive to decoding the PMAD instruction. Since Sun does not disclose a multiply-add instruction and since Sidwell does not disclose a sum of absolute differences, neither of the references discloses or suggests an execution unit to perform an operation initiated by the decode unit responsive to decoding the PSAD instruction and an operation initiated by the decode unit responsive to decoding the PMAD instruction.

Sidwell admits that the multiply-add unit is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied).

Since Sidwell teaches that the multiply-add execution unit 76 could perform only the operations initiated by the decode unit responsive to decoding a packed multiply-add instruction, Sidwell teaches away from the multiply-add execution unit 76 performing a first operation initiated by the decode unit responsive to decoding the PSAD instruction.

Conspicuously, Sidwell does not even share the multiplier functionality of the multiply-add unit 76 (used for muladd2ps) with the multiply instruction, mul2ps, of the obvious packed arithmetic unit 80 (col. 5, lines 37-43, col. 6, lines 19-22). Appellant respectfully submits that the alleged combination of references should not be considered obvious when Sidwell teaches away from precisely what appellant has done.

As cited above, the general rule applicable to a rejection based on a combination of references was stated in *Schaffer*, 108 USPQ at 328-329:

[References] may be combined for the purpose of showing that a claim is unparentable. However, they may not be combined indiscriminately, and to determine whether the combination of references is proper, the following criterion is often used: namely, whether the prior art suggests doing what an applicant has done. ... [It] is not enough for a valid rejection to view the prior art in retrospect once an applicant's disclosure is known. The art applied should be viewed by itself to see if it fairly disclosed doing what an applicant has done.

Moreover, without viewing the prior art in retrospect with the aid of appellant's disclosure, there is no suggestion in the cited references of an execution unit, as set forth by the instant claims at issue, to perform an operation initiated responsive to decoding the PSAD instruction and also an operation initiated responsive to decoding the PMAD instruction. Thus, appellant also submits that the presence in the claimed invention of an execution unit to perform operations initiated responsive to decoding both the PSAD instruction and also the PMAD instruction would be unexpected from the combined system of Sidwell and Sun.

Referring once again to Van Hook, appellant respectfully submits that if the pixel

distance computation circuit illustrated in Figure 9b could reasonably be expected to combine with Sidwell, it could not reasonably be expected to perform the multiply-add instruction of Sidwell, having no multipliers and the capacity of only 11 bits for full additions (Van Hook, Fig. 9b, 65, col. 11, line 3). Also, like Sidwell, Van Hook conspicuously includes only one instruction (PDIST) in the instruction group 206, for pixel distance unit 56 (Fig. 5, 56, col. 5, lines 16-17, Fig. 6c, 206).

Nor could Sidwell's multiply-add unit 76 reasonably be expected to perform the Sun's vis\_pdist() instruction. Sidwell's system provides no path for an accumulator input to packed arithmetic unit 6, for example, from result bus 56 or as a third source operand to packed multiply-add unit 76 (Figs. 1, 2, 4, and 6; col. 5, line 15 through col. 7, line 53). And as stated above, Sidwell does not even share the multiplier functionality of the multiply-add unit 76 (used for muladd2ps) with the multiply instruction, mul2ps, of the obvious packed arithmetic unit 80 (col. 5, lines 37-43, col. 6, lines 19-22).

Without viewing the prior art in retrospect with the aid of appellant's disclosure, the combined system of Sidwell and Sun could not reasonably be expected to include an execution unit to perform operations initiated responsive to decoding both the PSAD instruction and also the PMAD instruction. Therefore, appellant respectfully submits that no suggestion is provided by Sidwell or Sun for doing what appellant has done.

Because the execution unit used to perform the PMAD instruction can also perform the PADDH operation responsive to decoding both the PSAD instruction the utility of the PMAD execution unit may be increased by 100%, which is statistically significant. Of practical significance is that through relatively minor modifications to the existing execution unit a sum of absolute differences instruction can be supported with

only a negligible increase in circuit area.

Thus, the presence of an execution unit to perform operations initiated responsive to decoding both the PSAD instruction and also the PMAD instruction in the combined system of Sidwell and Sun would also be nonobvious.

Accordingly in light of the above arguments, Claims 16 and 36, are not obvious in view of the cited references.

### C. SECOND 35 U.S.C. § 103(a) REJECTIONS

Claims 21-24, 33-34 and 43-44 stand rejected under 35 USC § 103(a) as allegedly being unpatentable over US Patent 5,859,789 (Sidwell) in view of in view of Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun) and further in view of US Patent 5,721,697 (Lee).

#### 1. Claims 21-22, 33-34 and 43-44 Are Not Obvious.

With regard to Claims 21, 33 and 43, the Office Action states that it would have been obvious to combine Lee into a system of Sidwell and Sun to produce the features of the claims. Appellant respectfully disagrees.

In determining the scope and content of the cited references with regard to the instant claims at issue, appellant respectfully submits that Sidwell is directed to an arithmetic unit for packed arithmetic. The arithmetic unit 6 of Sidwell is comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19). Sidwell discloses that the obvious packed arithmetic unit 80 performs the addition, subtraction, comparison and multiplication of packed numbers (Figs. 4 and 5, col. 5, line 50 through col. 6, line 47). Sidwell discloses that the multiply-add unit 76 is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied). Sidwell does not disclose or suggest reuse of the multiply-add unit 76 even for performing packed multiply instructions (mul2ps).



Sun is directed to a set of visual instructions used primarily to write graphics and multimedia applications (p. 41, first paragraph). One of these instructions (the `vis_pdist()` instruction) accumulates the absolute values of differences into a destination accumulator (p. 87, last paragraph). Sun states that "the pixels are subtracted from one another, pair wise, and the absolute values of the differences are accumulated into *accum*" (p.87, 4.7.11, Description, first paragraph). Sun does not disclose or suggest a version of the `vis_pdist()` instruction that does not use an accumulation of prior absolute differences. Nor does Sun disclose or suggest a plurality of partial product selectors to insert an element of a plurality of elements of a packed data into and substituting for bit positions of one or more partial products and add the plurality of elements together.

Lee is directed to a multiplier that is modified to perform tree additions (Abstract). Lee aligns data from one input in partial products through use of second input value. Each bit of the second input value is set to zero except for a first subset of bits, starting with the low order bit which are set to one at intervals equal to a bit length of each addend (col. 1 lines 47-55, col. 4, line 44 through col. 5, line 2). Lee then generates control inputs to force to logic zero bit positions that do not correspond to the bit positions of an element to be added (col. 5, lines 3-5). In order to sum four 4-bit numbers, forty-eight (48) bit positions are forced to logic zero bit positions that do not correspond to the bit positions of the elements to be added together. (Table 6; cols. 6, lines 9-61). Lee does not disclose or suggest use of partial product selectors to insert the elements of a packed data into bit positions of the partial products to add the elements together.

Appellant respectfully points out some of the differences between the cited

references and the instant claims at issue. Claim 21, for example, sets forth:

21. (Previously Presented) The processor of Claim 16, wherein performing the first operation causes the execution unit to:
- produce a first plurality of partial products in a multiplier having a plurality of partial product selectors;
  - insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions;
  - and
  - add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit.

In addition to the limitations presented above with regard to Claim 16, Claim 21 sets forth that in performing the first operation responsive to decoding the packed sum of absolute differences instruction, the execution unit produces a plurality of partial products in a multiplier having a plurality of partial product selectors and inserts an element of a plurality of elements of a packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions and adding the plurality of elements together.

Neither Sidwell nor Sun disclose a plurality of partial product selectors to insert elements of a packed data into and substituting for bit positions of one or more partial products and adding the elements together.

Lee's method, on the other hand, generates control inputs to force to logic zero bit positions that do not correspond to the bit positions of an element to be added, rather than inserting elements of a packed data into and substituting for bit positions to be added as set forth in Claim 21 (col. 5, lines 3-5). Further, Lee aligns data from one input in partial products through use of another input value rather than using the partial product selectors corresponding to the bit positions to be added as set forth in Claim 21 (col. 1 lines 47-55).

Each bit of the second input value is set to zero except for a first subset of bits,

starting with the low order bit which are set to one at intervals equal to a bit length of each addend (col. 1 lines 47-55).

The `vis_pdist()` instruction of Sun already has three source operands, one of which is also the destination (p. 88, first paragraph). To perform the alignment in partial products as suggested by Lee a fourth source operand would be necessary. Sidwell's system provides no third path for source inputs to packed arithmetic unit 6, much less a fourth (Figs. 1, 2, 4, and 6; col. 5, line 15 through col. 7, line 53). Therefore Sidwell's system could not perform the alignment in partial products as disclosed by Lee for Sun's `vis_pdist()` instruction without substantial modification to the method or apparatus disclosed, and such modification, was not taught, suggested, or motivated by any of the cited references without viewing the prior art in retrospect with the aid of appellant's disclosure.

As cited above, the general rule applicable to a rejection based on a combination of references stated in *Schaffer*, 108 USPQ at 328-329 is that [it] is not enough for a valid rejection to view the prior art in retrospect once an applicant's disclosure is known. The art applied should be viewed by itself to see if it fairly disclosed doing what an applicant has done.

Without viewing the prior art in retrospect with the aid of appellant's disclosure, there is no suggestion in the cited references of producing a plurality of partial products in a multiplier having a plurality of partial product selectors and inserting an element of a plurality of elements of a packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions and adding the plurality of elements together.

Therefore, appellant respectfully submits that no suggestion is provided by Sidwell, Sun or Lee for doing what appellant has done.

Because Lee generates control inputs to force bit positions that do not correspond to the bit positions of an element to be added, forty-eight (48) bit positions are forced to logic zero in order to sum four 4-bit numbers (Table 6; cols. 6, lines 9-61). Thus circuitry at forty-eight (48) bit positions must be modified rather than circuitry at the sixteen (16) bit positions of the four 4-bit numbers being added. Such considerations are of practical significance.

As shown in the present application, with regard to Fig. 12 (1220 and 1221), only the sixty-four (64) bit positions being added need to be modified as apposed to the eighty (80) bit positions not corresponding to bit positions being added, which is of practical significance.



than aligning data from one input in partial products with a second input value and generating control inputs to force bit positions to zero that do not correspond to the bit positions being added, as was taught by Lee, would result in nine adjacent partial products (shown as shaded) being available for adding up to nine 8-bit values rather than of two. Thus the potential utilization of the multiplier circuitry may be increased by up to 350%, which is statistically significant.

Thus, producing a plurality of partial products in a multiplier having a plurality of partial product selectors and inserting an element of a plurality of elements of a packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions and adding the plurality of elements together in the combined system of Sidwell, Sun and Lee would be nonobvious.

Accordingly in light of the above arguments, Claims 21-22, 33-34 and 43-44 are not obvious in view of the cited references.

## 2. Claims 23-24 Are Not Obvious.

In determining the scope and content of the cited references with regard to the instant claims at issue, appellant respectfully submits that Sidwell is directed to an arithmetic unit for packed arithmetic. The arithmetic unit 6 of Sidwell is comprised of a collection of separate packed arithmetic execution units, each responsible for some subset of packed arithmetic instructions (Fig. 2, col. 5, lines 15-19). Sidwell discloses that the obvious packed arithmetic unit 80 performs the addition, subtraction, comparison and multiplication of packed numbers (Figs. 4 and 5, col. 5, line 50 through col. 6, line 47). Sidwell discloses that the multiply-add unit 76 is capable of executing a single instruction, the result of executing that instruction being to multiply together respective pairs of objects from two operands and to add together the results to provide a final result (col. 7, lines 21-31, emphasis supplied). Sidwell does not disclose or suggest producing a packed result having two distinct sums of products as a result of the multiply-add instruction. Nor does Sidwell disclose or suggest reuse of the multiply-add unit 76 even for performing packed multiply instructions (mul2ps).

Sun is directed to a set of visual instructions used primarily to write graphics and multimedia applications (p. 41, first paragraph). One of these instructions (the vis\_pdist() instruction) accumulates the absolute values of differences into a destination accumulator (p. 87, last paragraph). Sun states that "the pixels are subtracted from one another, pair wise, and the absolute values of the differences are accumulated into *accum*" (p.87, 4.7.11, Description, first paragraph). Sun does not disclose or suggest a version of the vis\_pdist() instruction that does not use an accumulation of prior absolute differences.

Nor does Sun disclose or suggest a plurality of partial product selectors to insert an element of a plurality of elements of a packed data into and substituting for bit positions of one or more partial products and add the plurality of elements together.

Lee is directed to a multiplier that is modified to perform tree additions (Abstract). Lee aligns data from one input in partial products through use of second input value. Each bit of the second input value is set to zero except for a first subset of bits, starting with the low order bit which are set to one at intervals equal to a bit length of each addend (col. 1 lines 47-55, col. 4, line 44 through col. 5, line 2). Lee then generates control inputs to force to logic zero bit positions that do not correspond to the bit positions of an element to be added (col. 5, lines 3-5). In order to sum four 4-bit numbers, forty-eight (48) bit positions are forced to logic zero bit positions that do not correspond to the bit positions of the elements to be added together. (Table 6; cols. 6, lines 9-61). Lee does not disclose or suggest use of partial product selectors to insert the elements of a packed data into bit positions of the partial products to add the elements together.

Appellant respectfully points out some of the differences between the cited references and the instant claims at issue. Claim 23, for example, sets forth:

23. A processor comprising:

a decode unit to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, and a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction; and  
an execution unit to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit;  
wherein performing the first operation causes the execution unit to:  
produce a first plurality of partial products in a multiplier having a



plurality of partial product selectors,

insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions, and

add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit;

and wherein performing the second operation causes the execution unit to:

produce a second plurality of partial products in the multiplier having the plurality of partial product selectors, the second plurality of partial products comprising four distinct sets of partial products including a first set of partial products corresponding to a first product for elements of the second set of packed data, a second set of partial products corresponding to a second product for elements of the second set of packed data, a third set of partial products corresponding to a third product for elements of the second set of packed data, and a fourth set of partial products corresponding to a fourth product for elements of the second set of packed data, and

add the first set of partial products together with the second set of partial products to produce a first distinct element of a packed result and add the third set of partial products together with the fourth set of partial products to produce a second distinct element of the packed result.

In addition to the limitations presented above with regard to Claim 16, Claim 23 sets forth that in performing the first operation responsive to decoding the packed sum of absolute differences instruction, the execution unit produces a plurality of partial products in a multiplier having a plurality of partial product selectors and inserts an element of a packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions.

As shown above with regard to Claim 21, neither Sidwell nor Sun disclose a plurality of partial product selectors to insert elements of a packed data into and substituting for bit positions of one or more partial products and adding the elements together.

Lee's method, on the other hand, generates control inputs to force to logic zero bit positions that do not correspond to the bit positions of an element to be added, rather than inserting elements of a packed data into and substituting for bit positions to be added as

set forth in Claim 21 (col. 5, lines 3-5). Further, Lee aligns data from one input in partial products through use of another input value rather than using the partial product selectors corresponding to the bit positions to be added as set forth in Claim 21 (col. 1 lines 47-55).

Appellant respectfully submits that no suggestion is provided by Sidwell, Sun or Lee for doing what appellant has done, and has established that the differences have both practical and statistical significance.

Thus, producing a plurality of partial products in a multiplier having a plurality of partial product selectors and inserting an element of a plurality of elements of a packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions and adding the plurality of elements together in the combined system of Sidwell, Sun and Lee would be nonobvious.

Claim 23 also sets forth that in performing the second operation responsive to decoding the packed multiply-add instruction, the execution unit produces four distinct sets of partial products including a first set corresponding to a first product, a second set corresponding to a second product, a third set corresponding to a third product, and a fourth set corresponding to a fourth product, and add the first set together with the second set of partial products to produce a first distinct element of a packed result and add the third set together with the fourth set of partial products to produce a second distinct element of the packed result.

The multiply-add of Sidwell does not add a first and second set of partial products and a third and fourth set of partial products produce a packed result having two distinct elements. Nor does Sidwell suggest another form of multiply-add instruction to produce

a packed result. Neither Sun nor Lee disclose or suggest a multiply-add instruction to add a first and second products and a third and fourth products produce a packed result.

Yet, the packed multiply-add disclosed by the present application to produce a packed result having two distinct elements may be useful in writing applications such as those considered by Sun or Sidwell or Lee. For example, one application known as alpha blending of images, performs computation of a pixel color value as follows:

$$\alpha_1 * s_1 + \alpha_2 * s_2$$

which would permit computation of two new pixel color values in parallel with one packed multiply-add instruction as set forth in Claim 23.

On the contrary, Sun discusses alpha blending using the VIS<sup>TM</sup> instructions and uses the fact that  $\alpha_2$  is equal to  $(1 - \alpha_1)$  to algebraically transform (p. 116, lines 24-25, minor corrections to Sun's algebra supplied):

$$(s_1 - s_2) * \alpha_1 + s_2.$$

Thus, Sun teaches away from computing the sum of two products as set forth in the packed multiply-add instruction of Claim 23, and teaches instead to perform a subtraction (vis\_fpsub16) a multiplication (vis\_fmuls16) and an addition (vis\_fpadd16) to compute four sums in parallel with three VIS instructions (e.g. see p. 118, lines 20-22 and 25-27).

As stated above, the general rule applicable to a rejection based on a combination of references stated in *Schaffer*, 108 USPQ at 328-329, is that it is not enough for a valid rejection to view the prior art in retrospect once an applicant's disclosure is known. The art applied should be viewed by itself to see if it fairly disclosed doing what an applicant has done.

Since neither Sidwell nor Lee disclose or suggest a multiply-add instruction to add

a first and second products and a third and fourth products produce a packed result having two distinct elements, and since Sun not only fails to disclose or suggest but teaches away from computing the sum of two products as set forth in the packed multiply-add instruction of Claim 23, appellant respectfully submits that the cited references do not fairly suggest, to one of skill in the art, a multiply-add instruction to add a first and second products and a third and fourth products produce a packed result having two distinct elements.

Therefore, appellant respectfully submits that no suggestion is provided by Sidwell, Sun or Lee for doing what appellant has done.

Because the multiply-add instruction adds the first and second products and the third and fourth products to produce a packed result having two distinct elements, two PMAD instructions could produce the four alpha blended sums that required three VIS instructions, resulting in a 33% reduction of instructions, which is statistically significant. Of practical significance is that the packed multiply-add to produce a packed result having two distinct elements provide support for applications such as alpha blending that require the sum of two products and not the sum of four products.

Thus, the presence of multiply-add instruction to add a first and second products and a third and fourth products produce a packed result having two distinct elements in the combined system of Sidwell, Sun and Lee would also be nonobvious.

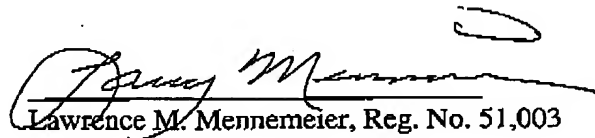
Accordingly in light of the above arguments, Claims 23-24 are not obvious in view of the cited references.

Conclusion

Appellant submits that all claims now pending are in condition for allowance. Such action is earnestly solicited at the earliest possible date. If there is a deficiency in fees, please charge our Deposit Acct. No. 02-2666.

Respectfully submitted,

Date: 8-10-05

  
Lawrence M. Mennemeier, Reg. No. 51,003

12400 Wilshire Boulevard  
Seventh Floor  
Los Angeles, CA 90025-1026  
(408) 720-8598

VIII. Claims Appendix: Claims Allowed and Involved in Appeal (Clean Copy)

1-15. (Cancelled)

16. (Previously Presented) A processor comprising:

a decode unit to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, and a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction; and

an execution unit to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit.

17. (Previously Presented) The processor of Claim 16, wherein the decode unit further decodes a plurality of instructions of a PENTIUM microprocessor instruction set.

18. (Previously Presented) The processor of Claim 16, wherein the first set of operations comprises:

a packed subtract and write carry (PSBWC) operation;  
a packed absolute value and read carry (PABSRC) operation; and  
a packed add horizontal (PADDH) operation.

19-20. (Cancelled)

21. (Previously Presented) The processor of Claim 16, wherein performing the first operation causes the execution unit to:

produce a first plurality of partial products in a multiplier having a plurality of partial product selectors;

insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions; and

add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit.

22. (Previously Presented) The processor of Claim 21, wherein performing the first operation further causes the execution unit to:

shift the first result to produce a second result having a least significant bit position and to align the least significant bit of the field with the least significant bit position of the second result.

23. (Previously Presented) A processor comprising:

a decode unit to decode a plurality of packed data instructions including a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, and a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a first set of operations on the first set of packed data responsive to decoding the PSAD instruction and to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction; and

an execution unit to perform a first operation of the first set of operations initiated by the decode unit and to perform a second operation of the second set of operations initiated by the decode unit;

wherein performing the first operation causes the execution unit to:

produce a first plurality of partial products in a multiplier having a plurality of

partial product selectors,

insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions, and

add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit;

and wherein performing the second operation causes the execution unit to:

produce a second plurality of partial products in the multiplier having the plurality of partial product selectors, the second plurality of partial products comprising four distinct sets of partial products including a first set of partial products corresponding to a first product for elements of the second set of packed data, a second set of partial products corresponding to a second product for elements of the second set of packed data, a third set of partial products corresponding to a third product for elements of the second set of packed data, and a fourth set of partial products corresponding to a fourth product for elements of the second set of packed data; and

add the first set of partial products together with the second set of partial products to produce a first distinct element of a packed result and add the third set of partial products together with the fourth set of partial products to produce a second distinct element of the packed result.

24. (Previously Presented) The processor of Claim 23, wherein the second format identifies the second set of packed data as packed words.

25. (Cancelled)

26. (Previously Presented) A processor to execute instructions of the PENTIUM microprocessor instruction set, the processor comprising:

decode logic to decode a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, said decode logic to initiate a



first set of operations on the first set of packed data responsive to decoding the PSAD instruction;

execution logic to perform a first operation of the first set of operations initiated by the decode logic; and

a bus to provide the first set of packed data to the execution logic for performing of the first operation.

27. (Previously Presented) The processor of Claim 26, wherein the decode logic comprises a look-up table.

28. (Previously Presented) The processor of Claim 26, wherein the decode logic comprises integrated circuitry.

29. (Previously Presented) The processor of Claim 28, wherein the decode logic further comprises executable operations.

30. (Previously Presented) The processor of Claim 29, wherein the decode logic comprises:

a packed subtract and write carry (PSBWC) operation;

a packed absolute value and read carry (PABSRC) operation; and

a packed add horizontal (PADDH) operation.

31. (Previously Presented) The processor of Claim 26, wherein the first format identifies the first set of packed data as packed bytes.

32. (Cancelled)

33. (Previously Presented) The processor of Claim 26, wherein performing the first operation causes the execution logic to:

produce a first plurality of partial products in a multiplier having a plurality of

partial product selectors;

insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions; and

add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit.

34. (Previously Presented) The processor of Claim 33, wherein performing the first operation further causes the execution logic to:

shift the first result to produce a second result having a least significant bit position and to align the least significant bit of the field with the least significant bit position of the second result.

35. (Previously Presented) The processor of Claim 26, the decode unit to decode a packed multiply-add (PMAD) instruction having a second format to identify a second set of packed data, said decode unit to initiate a second set of operations on the second set of packed data responsive to decoding the PMAD instruction.

36. (Previously Presented) The processor of Claim 35, execution unit to perform a second operation of the second set of operations initiated by the decode unit.

37. (Previously Presented) The processor of Claim 35, wherein the second format identifies the second set of packed data as packed words.

38. (Cancelled)

39. (Previously Presented) A processor comprising:

decode logic to decode a packed sum of absolute differences (PSAD) instruction having a first format to identify a first set of packed data, said decode logic to initiate a

first set of operations on the first set of packed data responsive to decoding the PSAD instruction, the first set of operations comprising:

- a packed subtract and write carry (PSUBWC) operation;
  - a packed absolute value and read carry (PABSRC) operation; and
  - a packed add horizontal (PADDH) operation; and
- execution logic to perform the first set of operations initiated by the decode logic.

40. (Previously Presented) The processor of Claim 39, wherein the first format identifies the first set of packed data as packed bytes.

41. (Previously Presented) The processor of Claim 39, wherein performing the PSUBWC operation causes the execution logic to:

- subtract one of a plurality of elements of a first packed data of the first set of packed data from a corresponding one of a plurality of elements of a second packed data of the first set of packed data to produce a first result having a plurality of difference elements and a plurality of sign indicators; and

- store the plurality of difference elements and the plurality of sign indicators.

42. (Previously Presented) The processor of Claim 39, wherein performing the PABSRC operation causes the execution logic to:

- receive a plurality of difference elements and a plurality of sign indicators;
- produce a result data having a plurality of absolute value elements, each absolute value element produced by

- (a) subtracting one of the plurality of difference elements from a corresponding constant value if the sign indicator corresponding to that element is in a first state, or

- (b) adding one of the plurality of difference elements to a corresponding constant value if the sign indicator corresponding to that element is in a second state.

43. (Previously Presented) The processor of Claim 39, wherein performing the PADDH operation causes the execution logic to:

produce a first plurality of partial products in a multiplier having a plurality of partial product selectors;

insert an element of a first plurality of elements of a first packed data into and substituting for bit positions of one or more of the first plurality of partial products by using partial product selectors corresponding to the bit positions; and

add the first plurality of elements together to produce a first result including a field comprising a sum of the first plurality of elements, said field having a least significant bit.

44. (Previously Presented) The processor of Claim 43, wherein performing the PADDH operation further causes the execution logic to:

shift the first result to produce a second result having a least significant bit position and to align the least significant bit of the field with the least significant bit position of the second result.

IX. Evidence Appendix: Copies of Evidence Relied Upon by Appellant

Exhibit A

- i. The "Pentium® Processor Family Developer's Manual, Vol. 3: Architecture and Programming Manual," 1995, pp. 25-165 and 25-166, cited by the Examiner in the Office Action (8.5) as extrinsic evidence for a common knowledge of the PENTIUM microprocessor instruction set by one of ordinary skill in the art.

The above cited reference was entered in the record by the examiner with the Office Action mailed on January 10, 2005.

- ii. A November 1997 article by Eric Traut from BYTE, which discusses a Macintosh application that employs a "Pentium instruction-set emulator, complete with MMX<sup>™</sup> instructions."
- iii. A definition of AMD from word1Q.com, which explains (in the History section, paragraph 7) that at some time about one year after AMD purchased NexGen in 1996, "the K6 [processor] translated the Pentium compatible x86 instruction set to RISC-like micro-instructions."
- iv. John Savill's FAQ (Frequently Asked Questions) for Windows web page, dated September 3, 1999, which asks, "Do I really need 166Mhz Pentium processors to run SQL Server 7.0?" The answer given states, "No. But you DO need a 100% PENTIUM compatible chip - which rules out some Cyrix and IBM processors." The page further explains (in paragraph 3) that, "speed of the processor doesn't matter as long as it runs the full pentium instruction set."
- v. A current product description of a single-board computer from SBS technologies, which includes a "Pentium compatible Geode GX1 processor."
- vi. A Department of Energy (hq.doe.gov) description of the Hardware & System Requirements for Microsoft Windows 2000 and Microsoft Office 2000 by IT Standards Manager, Carol Blackston, requiring a "133 MHz or higher Pentium-compatible CPU" for Windows 2000 Professional, a 166 MHz Pentium-compatible CPU or higher" for Office 2000 Premium, and a "75 MHz Pentium-compatible CPU or higher" for Office 2000 Professional or Office 2000 Standard.
- vii. Microsoft requirements for a Microsoft Operations Manager Server, a Database Server, a Reporting Server, or a SQL Server 2000 Reporting Services Server, listed as a "PC with 550 MHz or higher Pentium-compatible," an Administrator and Operator Console, listed as a "PC with 500 MHz or higher Pentium-compatible," and a Managed Computer, listed as a "PC with 200 MHz or higher Pentium-compatible."
- viii. An article by Taran Rampersad from the Free Software Consortium (FSC) dated March 26, 2004, describing the basic system requirements of OpenOffice under Windows (98, NT, 2000, XP) including a "Pentium-compatible PC."
- ix. An article by Thomas Latuske posted June 8, 2004, describing two ways to retrieve the processor-speed and stating (in paragraph 1) that, "If you want to use the function to calculate the speed (frequency), you have to use it with a Pentium instruction set compatible processor."

The above cited references were entered in the record by the examiner with the Declaration submitted under 37 CFR §1.132 on October 11, 2004.

x. US Patent 5,859,789 (Sidwell)

The above cited reference was entered in the record by the examiner with the Office Action mailed on August 19, 2003.

xi. Visual Instruction Set (VIS <sup>TM</sup>) User's Guide, Sun Microsystems, March 1997 (Sun)  
xii. US Patent 5,721,697 (Lee).

The above cited references were entered in the record by the examiner with the Information Disclosure filed by appellant on November 12, 2001.

xiii. US Patent 5,938,756 (Van Hook).

The above cited reference was entered in the record by the examiner with the Information Disclosure filed by appellant on November 26, 2001.

42390P5943C

A-i

THIS PAGE BLANK (USPTO)

42390P5943C



# **Pentium® Processor Family Developer's Manual**

## **Volume 3: Architecture and Programming Manual**

**NOTE:** The *Pentium® Processor Family Developer's Manual* consists of three books: *Pentium® Processor* Order Number 241428; the *82496/82497/82498 Cache Controller and 82491/82492/82493 Cache SRAM*, Order Number 241429; and the *Architecture and Programming Manual*, Order Number 241430. Please refer to all three volumes when evaluating your design needs.

1995



Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

The Pentium® processor may contain design defects or errors known as errata. Current characterized errata are available on request.

\*Third-party brands and names are the property of their respective owners.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation  
P.O. Box 7641  
ML Prospect, IL 60056-7641

or call 1-800-879-4683

COPYRIGHT © INTEL CORPORATION 1998



## INSTRUCTION SET

**IMUL—Signed Multiply**

Opcode	Instruction	Clocks	Description
F6 /S	IMUL <i>rm8</i>	11	AX ← AL * <i>rm</i> byte
F7 /S	IMUL <i>rm16</i>	11	DX:AX ← AX * <i>rm</i> word
F7 /S	IMUL <i>rm32</i>	10	EDX:EAX ← EAX * <i>rm</i> dword
0F AF /r	IMUL <i>r16,rm16</i>	10	word register ← word register * <i>rm</i> word
0F AF /r	IMUL <i>r32,rm32</i>	10	dword register ← dword register * <i>rm</i> dword
6B /r /b	IMUL <i>r16,rm16,imm8</i>	10	word register ← <i>rm16</i> * sign-extended immediate byte
6B /r /b	IMUL <i>r32,rm32,imm8</i>	10	dword register ← <i>rm32</i> * sign-extended immediate byte
6B /r /b	IMUL <i>r16,imm8</i>	10	word register ← word register * sign-extended immediate byte
6B /r /b	IMUL <i>r32,imm8</i>	10	dword register ← dword register * sign-extended immediate byte
69 /r /w	IMUL <i>r16,r/m16,imm16</i>	10	word register ← <i>rm16</i> * immediate word
69 /r /d	IMUL <i>r32,r/m32,imm32</i>	10	dword register ← <i>rm32</i> * immediate dword
69 /r /w	IMUL <i>r16,imm16</i>	10	word register ← <i>rm16</i> * immediate word
69 /r /d	IMUL <i>r32,imm32</i>	10	dword register ← <i>rm32</i> * immediate dword

**Operation**

result ← multiplicand \* multiplier;

**Description**

The IMUL instruction performs signed multiplication. Some forms of the instruction use implicit register operands. The operand combinations for all forms of the instruction are shown in the "Description" column above.

The IMUL instruction clears the OF and CF flags under the following conditions (otherwise the CF and OF flags are set):

Instruction Form	Condition for Clearing CF and OF
<i>r/m8</i>	AL = sign-extend of AL to 16 bits
<i>r/m16</i>	AX = sign-extend of AX to 32 bits
<i>r/m32</i>	EDX:EAX = sign-extend of EAX to 32 bits
<i>r16,r/m16</i>	Result exactly fits within r16
<i>r32,r/m32</i>	Result exactly fits within r32
<i>r16,rm16,imm16</i>	Result exactly fits within r16
<i>r32,rm32,imm32</i>	Result exactly fits within r32

**Flags Affected**

The OF and CF flags as described in the table in the "Description" section above; the SF, ZF, AF, and PF flags are undefined.

**INSTRUCTION SET****Protected Mode Exceptions**

#GP(0) for an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments; #SS(0) for an illegal address in the SS segment; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3.

**Real Address Mode Exceptions**

Interrupt 13 if any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.

**Virtual 8086 Mode Exceptions**

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference if the current privilege level is 3.

**Notes**

When using the accumulator forms (IMUL r/m8, IMUL r/m16, or IMUL r/m32), the result of the multiplication is available even if the overflow flag is set because the result is twice the size of the multiplicand and multiplier. This is large enough to handle any possible result.

A-ii

**THIS PAGE BLANK (USPTO)**

42390P5943C

**Archives**

Columns

Features

Print Archives  
1994-1998**Special**

BYTE Digest

Michael Abrash's  
*Graphics Programming  
Black Book*

101 Perl Articles

**About Us**How to Access  
BYTE.com

Write to BYTE.com

Advertise with  
BYTE.com**Newsletter**Free E-mail  
Newsletter from  
BYTE.com

your email here



► SEARCH:

Jump to...

[HOME](#) [ABOUT US](#) [ARCHIVES](#) [CONTACT US](#) [ADVERTISE](#) [REGISTER](#)[ARTICLES](#) [BYTEMARKS](#) [FACTS](#)**Building the Virtual PC****November 1997 / Core Technologies / Building the Virtual PC**

***A software emulator shows that the PowerPC can emulate another computer, down to its very hardware.***

**Eric Traut**

Development of Virtual PC -- Connectix Corporation's Macintosh application that emulates a PC and its peripherals -- began almost two years ago, in October 1995. The goal from the beginning was to create a fully Intel-compatible PC in software. The effort centered around a core Pentium instruction-set emulator, complete with MMX instructions. True PC emulation also required the reverse-engineering and development of a dozen other PC motherboard devices, including modern peripherals such as an accelerated SVGA card, an Ethernet controller, a Sound Blaster Pro sound card, IDE/ATAPI controller, and PCI bridge interface. This strategy of hardware-level emulation resulted in an application that allows Macintosh users to run not only Windows programs and DOS games but several x86-based OSes, including Windows 95, NT, and NeXT OpenStep.

**Pentium Emulation**

The heart of Virtual PC is the Pentium recompiling emulator, a sophisticated piece of software written entirely in hand-coded PowerPC assembly language. Its job is to translate Pentium instruction sequences into a set of optimized PowerPC instructions that perform the same operation. Translation occurs on a "basic-block" basis, where

a basic block consists of a sequence of decoded x86 instructions. Basic blocks end on an instruction that abruptly changes the flow of execution (typically a jump, call, or return-from-subroutine instruction). As the recompiler decodes x86 instructions, it analyzes them for "condition code" usage. Finally, it generates a block of PowerPC code that accomplishes the same task. For more details on this process, see "Virtual PC Operation".

For purposes of speeding things up, the emulator employs the following tricks.

**Translation cache:** Even though written in PowerPC assembly language, the translator still requires substantial time to generate optimized instruction translations. To reduce this overhead, the emulator caches blocks of translated code.

**Interinstruction optimization:** Because the Pentium is a CISC processor, most instructions perform more than one operation. For example, the ADD instruction not only adds two values together, it also produces a number of condition-code flags that tell programs whether the addition produced a zero or negative result. Such codes are used, for example, to determine if a program performs a conditional jump. Most of the time these codes are ignored. The translator analyzes blocks of x86 instructions to determine which flags the program uses (if any). It then generates PowerPC code for those flags actually used. The first two listings in "Translated Code" show how one Pentium instruction translates into three PowerPC instructions, while three Pentium instructions can be optimized from nine into five PowerPC instructions.

**Address translation:** One of the most difficult Pentium features to emulate is its built-in memory management unit (MMU). This hardware translates linear (or logical) addresses into physical memory addresses. Operating systems use the MMU to implement virtual memory and memory protection. Because of the Pentium's small register file, about three in four Pentium instructions reference memory in one way or another. Each memory address potentially needs to be translated before the emulator loads from, or stores to, the referenced address. An MMU implemented in software would impose a high overhead, which would degrade performance. Luckily, this overhead can be avoided: The Connectix engineers were able to program the PowerPC's MMU to mimic the Pentium MMU's behavior, thus managing the address translations in hardware. The Pentium's memory page attributes can also be mirrored in the PowerPC's MMU. For example, if Virtual PC's emulated OS marks a memory page as write-protected, the page mappings are modified so the corresponding PowerPC page is write-protected.

**Segment bounds checking:** The Pentium architecture includes the archaic notion of memory segments. Every memory reference, such as instruction fetches, stack operations, loads, and stores, has an associated memory segment. When a segment's bounds are

exceeded, the Pentium's MMU generates a general protection fault (GPF). The OS uses GPFs for more than detecting bugs in applications: They enable a program to "thunk" down into privileged driver-level code not accessible at the application level. Therefore, the Pentium emulator must detect segment bound faults where appropriate. Although the PowerPC does not contain segmentation hardware akin to the Pentium, Connectix used PowerPC `trap` instructions to perform segment bounds checks with little or no overhead.

### Hardware Emulation

Besides the Pentium processor, a typical PC motherboard contains a dozen or so chips that work together concurrently. All these chips need to be emulated faithfully for compatibility. The Intel architecture provides an I/O address space that's used to access hardware outside of the CPU. You work with this "I/O space" through two instructions -- `IN` and `OUT`. When using these instructions, software must specify an I/O port (or address). Virtual PC routes I/O accesses to code modules that emulate each chip. For example, if Virtual PC encounters an `IN` instruction referencing port 0x21, it calls a routine in the interrupt-controller emulation module that returns the current interrupt mask. Similar module calls occur for every I/O space access, as the third listing in "Translated Code" shows.

Many of the extra chips on a PC motherboard control I/O devices such as the hard drive, CD-ROM, keyboard, and mouse. For compatibility with the Mac OS and all Macintosh hardware, Virtual PC performs all I/O through the standard Mac OS drivers. So, a request sent to the emulated PC's IDE controller to read a sector from the hard drive gets translated into a read operation that's sent to the Mac OS SCSI driver.

The most difficult hardware components to emulate involve precise timing. For example, sound is a real-time operation, and any timing perturbation results in clicks or pops as digitally sampled data fails to arrive on time. Because Virtual PC is hosted on the Mac OS (which gives time to other Mac programs running concurrently, as well as Virtual PC), and it needs to emulate several dozen PC chips in parallel, precise timing isn't always possible. Virtual PC compensates by placing the highest priority on tasks that directly affect the user, such as sound and video.

### Performance

Emulated systems are naturally going to be slower than real hardware. But Connectix engineers concentrated on tuning aspects of the emulated hardware required to run popular PC games and productivity applications at a usable performance level. This was especially challenging given that the PowerPC processor emulates not only the Pentium but all the other chips on a PC motherboard.

Performance of Virtual PC is also greatly affected by the host hardware system. The latest PowerPC processors with high clock rates

and large on-chip caches will run it best. The speed and size of the system's L2 cache is also critical because of the code expansion that occurs during the translation process.

While users will take a performance hit because this is an emulator, Virtual PC successfully emulates the entire PC at a very low level. PC programs -- applications, device drivers, and operating systems alike - cannot tell they are not running on actual PC hardware.

## Translated Code

### Translation of Single Pentium Instruction

Pentium instruction	PowerPC instructions
ADD EAX,20	li            rTemp1,20 addco.      PF,rTemp1,rEAX mr           rEAX,rPF

### Translation of Pentium Instruction Block

Pentium instructions	PowerPC instructions
ADD EAX,20	add            rEAX,rEAX,20
ADD EBX,30	add            rEBX,rEBX,30
ADD ECX,40	li            rTemp1,40 addco.      rPF,rTemp1,rECX mr           rECX,rPF

### Code Translation for Pentium I/O Instructions

Pentium instructions	PowerPC instructions
MOV AL,8	li            rAL,8
MOV DX,0x1F0	li            rDX,0x1F0
OUT DX,AL	bl            HandleIDEPortWrite
AD	
D DX,7	addi          rDX,rDX,7
IN AL,DX	bl            HandleIDEPortRead
RET	addi          rIP,rIP,8 b            DispatchToNextBlock

## Virtual PC Operation

[illustration link \(24 Kbytes\)](#)



**Eric Traut ( [traut@connectix.com](mailto:traut@connectix.com) ) is lead engineer for Virtual PC at Connectix. At Apple Computer, he wrote the 680x0 dynamic recompiling emulator for PowerPC-based Macs.**



BYTE.com

Page 5 of 6



Up Level



Previous



Next

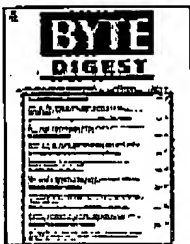
### Flexible C++

*Matthew Wilson*

My approach to software engineering is far more pragmatic than it is theoretical--and no language better exemplifies this than C++.

[more...](#)

### BYTE Digest



*BYTE Digest* editors every month analyze and evaluate the best articles from *Information Week*, *EE Times*, *Dr. Dobbs's Journal*, *Network Computing*, *Sys Admin*, and dozens of other CMP publications—bringing you critical news and information about wireless communication, computer security, software development, embedded systems, and more!

[Find out more](#)

**BYTE.com Store**

**BYTE CD-ROM**

NOW, on one CD-ROM, you can instantly access more than 8 years of BYTE.



**The Best of BYTE**  
**Volume 1:**  
**Programming**  
**Languages**

In this issue of *Best of BYTE*, we bring together some of the leading programming language designers and implementors...

.....  
Copyright © 2005 CMP Media LLC, Privacy Policy, Your California Privacy rights, Terms of Service  
Site comments: [webmaster@byte.com](mailto:webmaster@byte.com)  
SDMG Web Sites: [BYTE.com](http://BYTE.com), [C/C++ Users Journal](http://C/C++ Users Journal), [Dr. Dobbs's Journal](http://Dr. Dobbs's Journal), [MSDN Magazine](http://MSDN Magazine), [New Architect](http://New Architect), [SD Expo](http://SD Expo), [SD Magazine](http://SD Magazine), [Sys Admin](http://Sys Admin), [The Perl Journal](http://The Perl Journal), [UnixReview.com](http://UnixReview.com), [Windows Developer Network](http://Windows Developer Network)

A-iii

THIS PAGE BLANK (USPTO)

42390P5943C



## Definition of Amd

*For other possible meanings of AMD see AMD (disambiguation)*

**Advanced Micro Devices, Inc. (AMD)** (NYSE:

**AMD** (<http://www.nyse.com/about/listed/lcddata.html?ticker=AMD>)) is a manufacturer of integrated circuits based in Sunnyvale, California. It is the second-largest supplier of x86 compatible processors, and a leading supplier of non-volatile flash memory. It was founded in 1969 by a group of defectors from Fairchild Semiconductor, including the flashy Jerry Sanders. AMD's current president and CEO is Dr. Hector Ruiz.

AMD is best known for the Athlon, Opteron and Duron lines of x86-compatible processors. Their more general components have been found in early Apple computers and numerous other electronic devices.

### Contents

- 1 Financial information
- 2 History
- 3 AMD64
- 4 Geode
- 5 See also
- 6 External links

## Financial information

AMD is publicly traded at NYSE with the symbol AMD. Its market capitalization was around US\$8 billion at the end of 2004.

## History

**AMD**

The company started as a producer of logic chips in 1969 and entered the RAM chip business in 1975. That same year, it introduced a reverse-engineered clone of the Intel 8080 microprocessor. During this period, AMD also designed and produced a series of bit-slice processor elements (Am2900, Am29116, Am293xx) which were used in various minicomputer designs.

In February 1982, AMD signed a contract with Intel, becoming a licensed second-source manufacturer of the 8086 and 8088 processors. IBM wanted to use the Intel 8088 in its IBM PC, but IBM's policy at the time was to require at least two sources for its chips.

AMD later produced the 80286, or 286, under the same arrangement, but then Intel cancelled the agreement in 1986. The growing popularity of the PC clone market meant Intel could produce CPUs on its terms, rather than IBM's.

In 1991, AMD released the Am386, its clone of the later Intel 80386 processor. It took less than a year for AMD to sell a million units. AMD followed in 1993 with the Am486. Both sold at a significantly lower price than the Intel versions. Intel challenged AMD's right to produce these chips in court, but ultimately lost its case. The two competitors have had full cross-licensing agreements for patents and some copyrights from the very start: each partner can use the other's technological innovations without charge. AMD's 386DX-40 was very popular with smaller, independent clone manufacturers, and the Am486 was used by a number of large OEMs, including Compaq.

During this time, AMD attempted to embrace the perceived shift towards RISC with their own AMD 29K processor, and they attempted to diversify into graphics and audio devices as well as flash memory. While the AMD 29K survived as an embedded processor and AMD continues to make industry leading flash memory, AMD was not as successful with its other endeavours. AMD decided to switch gears and concentrate solely on Intel compatible microprocessors and flash memory. This put them in direct competition with Intel for x86 compatible processors and their flash memory secondary markets.

Their first completely in-house processor was the K5, launched very belatedly in 1995. The "K" was a reference to "Kryptonite". It was intended to compete directly with the Intel Pentium CPU, which had been released in 1993, but architecturally it had more in common with the newly-released Pentium Pro than the Pentium or Cyrix's 6x86. There were a number of problems however; a confusing naming system was employed, with some chips being represented by their true core speed, others with a PR number. More tellingly, the K5 couldn't match the 6x86's integer performance, nor the Pentium's FPU performance. This, combined with the large die size and the fact that the design scaled badly, doomed the K5 to near-total failure in the market place. To its credit, however, the K5 didn't suffer from the compatibility problems that the 6x86 did, and it didn't run as hot as Cyrix's chip.

Missing image  
AMD\_C8080A.jpg  
Early AMD 8080 Processor  
(AMD AM9080ADC /  
C8080A), 1977

In 1996, AMD purchased NexGen, Inc. and the rights to intellectual property behind their Nx series of x86 compatible processors. In a year, they reworked the Nx686 microarchitecture and branded it the K6. NexGen's original design had never made it to market. The redesign included a feedback dynamic instruction reordering mechanism, and MMX instructions. Most importantly AMD made it pin-compatible with Intel's Pentium, enabling it to be used in the widely available "Socket 7" based motherboards. Like the Nx686 and Nx586 before it, the K6 translated the Pentium compatible x86 instruction set to RISC-like micro-instructions. In the following year, AMD released the K6-2 which added a set of floating point multimedia instructions called 3DNow!, as well as a new socket standard called "Super Socket 7" both of which delivered enhanced performance.

In January 1999, the final iteration of the K6-x series, the 450 MHz K6-III, was extremely competitive with Intel's top of the line chips. This chip was essentially a K6-2 with 256 kilobytes of full-speed level 2 cache integrated into the core and a better branch prediction unit. While it matched (generally beating) the Pentium II/III in integer operations, the FPU was a non-pipelined serial design and could not compete with Intel's more advanced FPU architecture. Although 3DNow! could theoretically compensate for this weakness, few game developers made use of it, the most notable exception being id Software's Quake 2.

Throughout its lifetime, the K6 processor came close, but never quite seemed to equal the performance of processor offerings from Intel. Furthermore, the motherboards that worked with the K6 were of varying quality, and AMD had process manufacturing difficulties which affected some shipments. AMD gained a reputation of making a somewhat slower and less reliable "x86 clone" even though the performance difference was slight and the best K6 compatible motherboards were very reliable. This forced AMD into a position of selling their K6 processors at a substantial discount versus Intel's P6 core based processors, the Pentium II and the Pentium III. Intel responded to AMD's lower prices with the "Celeron" version of their Pentiums which were cheaper and slower in a partially successful attempt to capture marketshare.

In August of 1999, AMD released the Athlon (K7) processor. The Athlon had an advanced micro-architecture geared towards overall performance. The timing of the release of this processor put it at a great performance advantage versus Intel's P6 core based processors (which culminated as their mainline processor in the Pentium III.) The Intel P6 core was nearing the end of its life-cycle, while the Athlon was just getting started. Objectively, the Athlon had higher "per clock" architectural performance versus the comparable Intel P6 core based parts, as well as higher frequencies. AMD announced a 1GHz Athlon in early March 2000 and delivered them in that same month. Intel also announced a 1GHz Pentium a few days later, but did not ship them in significant volume until June of that year.

AMD also worked hard to increase the reliability and performance of motherboards for the Athlon. They also improved the discipline and predictability of their manufacturing process. AMD also released a second line of processors based

on the Athlon core called the Duron which was a slower and cheaper version of the processor aimed at competing against the still-shipping Celeron processor, providing some insulation for the Athlon against AMD's prior reputation for only making cheaper and slower "Intel clones". The combination of these technical and marketing successes did much to repair and bolster AMD's reputation for making high-performance CPUs that shipped and worked reliably. AMD continued to undercut Intel on price which helped them establish up to 20% market share.

In 2001, Intel released the Pentium 4 architecture (code-named Willamette) which had a radically different microarchitecture than the Athlon or the P6 cores. While sporting a dramatically higher clock rate, the per-clock architectural performance of the Pentium 4 appears to be much slower than the Athlon or even Intel's own P6 core based processors. This lead some to believe that the Pentium 4 had higher performance because of its higher clock rate, despite benchmark performance.

AMD responded with a new K7 core (code-named Palomino) which had superior memory pre-fetching mechanisms, SSE (a set of floating-point extensions first featured on the Pentium III), an on-chip L2 cache and also re-branded them based on a PR rating which would approximately project the clock rate relative performance of these new Athlons versus the earliest versions of the Athlon. The net effect of this was for the Model numbers to be more comparable to the Pentium 4's actual clock rate. For AMD processors of a given Model number, the comparable Pentium 4 by corresponding clock rate showed rough parity on performance in a wide variety of benchmarks.

Intel countered AMD and its Athlon by ramping the Pentium 4 clock rate aggressively in its early lifetime, just as the Athlon was nearing its end of life, giving it a brief period of performance dominance. AMD responded with the "Thoroughbred" Athlon XP, essentially a 130 nanometre version of the Palomino. AMD later introduced the "Barton" core, which increased the L2 cache to 512 KIB.

## AMD64

AMD's future strategy is shown with the 64-bit AMD64 "Hammer" architecture (a.k.a. K8), based on the AMD Opteron. Whilst retaining support for the traditional x86 instruction set, the Hammer's native 64-bit mode is unique to AMD processors and incompatible with the IA-64 architecture used in Intel's Itanium processor (Intel has since announced an extension to their Xeon CPU based on and compatible with AMD64 known as EM64T). As a relatively straightforward extension and cleanup of the basic x86 architecture, from a technical perspective AMD's conservative approach looks likely to produce, at least initially, better price-performance than the Itanium and its successors.

This also gives AMD a marketing advantage in that it can leverage its ordinary 32-bit x86 processor market to naturally upgrade and adopt its 64-bit processors without introducing risk to the existing software infrastructure. The potential

for this processor to compete with Itanium head on in its intended markets (high-end 64 bit servers) remains unclear.

AMD released its first AMD64 processor (K8), the Opteron, in March 2003. The Opteron is designed for workstation and server systems, including those containing more than one processor. However, Cray Inc. announced that it was going to use the Opteron as the basis for a top of the line super computer called "Red Storm", indicating that there seemed to be no limit for what sort of applications the Opteron could be used for. AMD then released Athlon 64 and Athlon 64 FX in September 2003 based on the same core architecture, which most benchmarks indicated both as performing equal or better than the Pentium 4. The Pentium 4 was previously a boon to Athlon 64, and had retained its performance advantage in streaming media processing applications. As of November 2004, there had been the 2700+, 2800+, 3000+, 3200+, 3400+, 3700+, 3800+, and 4000+ for the Athlon 64.

The AMD Athlon FX and the Opteron use a different numbering system. The Opteron includes three series, the 100's, 200's, and the 800's, each of which are meant for different types of servers and workstations. 100 Series Opterons are intended for Workstation, and uniprocessor configuration. The 200 series are intended for workstation or server use, and are not qualified to use more than 2 processors in the same system. 800 Series Opterons are used in 4 or 8 way (CPU) servers which anchor some of the most powerful computers built using Opteron. The Athlon 64 FX uses numbers going up, starting with 51. The three models available for the high-end Athlon FX include the FX-51, FX-53 and the FX-55. The recent release of the Athlon 64 FX-55 has shown that up against the Pentium 4 Extreme Edition, you can't lose if you have the cash to spend on one, as the Pentium 4 Extreme Edition loses out on many categories that it previously held the crown in (like Media Encoding or Adobe Premiere). AMD thus regains the crown in those areas once again. The Hammer core is very similar to the Athlon in basic microarchitecture, but includes 4 major differences:

1. The inclusion of the AMD64 instruction set;
2. A built-in DDR memory controller;
3. The HyperTransport point-to-point-bus.
4. Support for the NX, or No-Execute bit, which serves to act as a hardware buffer against viruses and other software exploits.

These improve both the capabilities and performance of the Hammer versus the K7 Athlon.

## Geode

In August 2003 AMD also purchased the Geode business (originally the Cyrix MediaGX) from National Semiconductor to augment its existing line of embedded x86 processor products. During the 2nd Quarter of 2004, it planned to launch new low-power Geodes with speeds just over 1 GHz.



## See also

- Turion
- List of AMD microprocessors
- List of AMD CPU slots and sockets
- Jerry Sanders

## External links

- AMD Corporate Website (<http://www.amd.com/>)
- NerdyPC ([http://nerdypc.wikinerdz.org/index.php/Advanced\\_Micro\\_Devices,\\_Inc.\\_article](http://nerdypc.wikinerdz.org/index.php/Advanced_Micro_Devices,_Inc._article))
- Cpu-collection.de (<http://www.cpu-collection.de/?10=co&11=AMD>) - AMD processor images and descriptions
- Yahoo! (<http://biz.yahoo.com/fc/10/10037.html>) - Advanced Micro Devices, Inc. Company Profile

de:Advanced Micro Devices es:AMD eo:AMD fr:Advanced Micro Devices he:AMD hu:AMD nl:Advanced Micro Devices ja:AMD no:Advanced Micro Devices pl:Advanced Micro Devices pt:AMD sk:AMD sl:AMD sr:Advanced Micro Devices sv:AMD vi:AMD zh:AMD

## Encyclopedia Index: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

<b>Definition</b> quotes, news, charts, and brokers. Definition.	<b>Online Dictionary</b> Look up dictionary terms Instantly! Free reference dictionary toolbar.	<b>AMD64 Processors</b> Get Smarter, Faster Computing Performance w/AMD64. Learn More!	<b>Britannica 2005</b> Encyclopedia Britannica new 2005 now accepting orders. free shipping
--	--	---	--

Ads by Google

All text is available under the terms of the GNU Free Documentation License. Privacy Policy :: Terms of Use :: Contact Us :: About Us  
This article is licensed under the GNU Free Documentation License. It uses material from the Wikipedia article "Amd". The list of authors can be found here.

A-iv

THIS PAGE BLANK (USPTO)

42390P5943C

Print - Do I really need 166Mhz Pentium processors to run SQL Server 7.0?

Page 1 of 3

See the European Edition [Subscribe](#)

Windows TPro [Keyword Search](#) [GO](#) [InstantDoc ID](#) [GO](#) [Advanced Search](#)

Connecting the IT Community [Home](#) [Forums](#) [Events](#) [Topics](#) [Publications](#) [Articles](#) [Blogs](#) [IT Products & S](#)

**MEET SECURITY THREATS**Find the tools and guidance you need  
for a well-guarded network ▶

Micro

[Free Tools & Updates](#)[ISA Server 2004](#)[Windows XP](#)

[SQL]

**Do I really need 166Mhz Pentium processors to run SQL Server 7.0?**

Neil Pike  
InstantDoc #14153  
September 3, 1999



A. No. But you DO need a 100% PENTIUM compatible chip - which rules out some Cyrix and IBM processors. The only way around this is for the chip vendor to offer a micro-code upgrade. (Some non-Intel chips say they are pentiums, but in fact only implement the 486 chip-set).

The following quote is from Cyrix - "Recently an issue with SQL Server 7.0 has been discovered with the non-MMX Media GX and the 6x86 processors. A fix for this issue can be obtained from Cyrix technical support at: [tech\\_support@cyrix.com](mailto:tech_support@cyrix.com)"

The actual speed of the processor doesn't matter as long as it runs the

full pentium instruction set - it needs to support CMPXCHG8B (Compare and Exchange 8 bytes) and RDTSC (Read Time-Stamp counter) instructions. Microsoft have made this a requirement because it is the minimum spec machine that they have developed/tested with - which is ok if you get most of your equipment donated/loaned/replaced by hardware companies free of charge, but this isn't the case with most businesses!

As long as the server previously ran SQL 6.5 (and is 100% PENTIUM compatible) you should find that it will run SQL 7.0 and will offer significant performance improvements, so don't upgrade hardware for the sake of it.

The following quote is from Microsoft Product Support Services :-

"When using SQL Server v7.0, Microsoft recommends a processor speed of 166Mhz or higher for server machines. Our extensive testing of the product has been done on machines of this calibre and we believe customers will get a better price performance with the product when used in this configuration. Microsoft will support SQL Server v7.0 when run on server machines with slower processors. However customers should recognise that if our findings are that major problems can be eliminated by using faster processors we will continue to recommend, and in some cases may require, compliance with this suggestion."

The reason for this caveat is that some of the decisions the optimiser makes on a 166Mhz pentium may not make so much sense on a 60Mhz pentium - i.e. the extra cpu time a 60Mhz part needed may mean that a non-optimal plan had been chosen.

#### Windows IT Pro Marketplace

##### Argent versus MOM 2005

Download Argent Versus Microsoft Operations Manager 2005

##### Exchange Preventative Maintenance

Find out how -- download your FREE Essential Guide now!

##### Tech jobs at Dice

Search 65K+ new IT jobs daily--Tech expert jobs at top companies!

##### Do you need an Email Compliance Policy?

Where do you start - download the whitepaper now.

##### A new dimension in IT

Access to KVM, serial console, and power control under one platform.

##### Test Your Security Configuration

Identify and fix vulnerabilities - download the free whitepaper now!

##### Backup Windows Servers - Free Trial

Cut server backup costs in half with Backup for Workgroups!

#### Featured Links

##### Learn To Migrate to SQL Server 2005

SQL Server 2005 Roadshow coming to your city. Register now!

##### AUGUST SPECIAL-Get 44%off Windows IT Pro

Sign up now and start getting quick answers to your Windows questions

##### 15-Minute Failover Solution for Exchange

Attend and win a \$50 gift certificate to Best Buy

##### Get 2 Free Issues of SQL Server Magazine

Put SQL Server Magazine to the test - you won't want to miss a single issue!

##### Best Practices for the Mobile Enterprise

Identify the key security considerations for wireless mobility

##### Compliance Vs Recovery - New Web Seminar

Integrate your compliance system with backup and recovery

##### Windows Innovators Contest 2005

Deadline Extended - Win a trip to Exchange Connections!

#### Ads by Google

##### Stylish Print

Order Print Online. Fun Designs and Cheap Prices.  
www.GiftChecks.com

##### We Print Everything

Top Quality, Competitive Pricing, Personal Service! (212) 967-8900  
venusprinting.brobsource.com

##### MCSE Exam Questions

All actual screen questions 100% pass at first test  
www.examkiller.net

##### Articles

Compare - Articles OnLine M: Analysis  
www.BlastSeek.com

#### Our Other Websites:

CertTutor | Connected Home | JSI FAQ | IT Library/eBooks | SuperSite | Windows FAQ | WinInfo News | Europe Edition | MSD2D

[Home](#) | [Subscribe / Register](#) | [About Us](#) | [Contact Us / Customer Service](#) | [Affiliates / Licensing](#) | [Press Room](#) | [Media Kit](#) | [RSS](#)

Windows IT Pro is a Division of Penton Media Inc.

Print - Do I really need 166Mhz Pentium processors to run SQL Server 7.0?

Page 3 of 3

Copyright © 2005 Penton Media, Inc., All rights reserved. [Legal](#) | [Privacy Policy](#)



A-v

42390P5943C

EC6 - Single board computer based on Geode GX1 processor with PC104+ expansion slot

Site Map Investor Relations Careers Support Contact Us

Jump to a product

**SBS**  
Technologies.

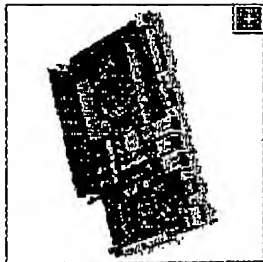
product search

SYSTEMS INTEGRATION  
SINGLE BOARD COMPUTERS  
PC104+ EXPANSION  
PC104+ COMMUNICATIONS  
PRESS ROOM  
COMPANY INFO

Home : Single Board Computers & CPUs : Industrial PC : EC6

## EC6

Single board computer based on Geode GX1 pro



- Geode GX1 (200 - 333 MHz) processor
- Dual 10BaseT/100BaseTX Ethernet interfaces
- Up to 512 MB SDRAM
- 128 KB nvSRAM

## FEATURED PRODUCT

### CA3



The Ultra Low Voltage Intel® Celeron® 650 processor makes the CA3 ideal for power-critical applications. It is based on the same core as the Intel® Pentium® III Processor and is optimized for high performance with low power consumption.

EC6 is a stand-alone all-in-one single board computer offering scalable processing power with low power consumption. EC6 features a low-power, Pentium-compatible Geode GX1 processor (200 - 333 MHz), compact design, integrated power supply filter, fanless cooling and PC104+ expansion slots for modular expansion. The cost-efficient EC6's rich feature set allows an extended temperature range of -40 to +70 C to accommodate fanless operation in harsh environments, and has low power requirements to eliminate expensive power supplies and enable use in mobile applications. EC6 comes with flexible DRAM configurations and an extensive array of peripherals, including video interface, IDE interface, and a PC104+ interface to enable custom extensions such as Interbus-S, Profibus, and Device-Net. EC6 also provides cost-efficient customization options to meet unique application requirements. Operating system support is available for Windows NT / 2000 / XP, VxWorks, MS-DOS and others.

VIEW DATASHEET  
SUPPORT

### Related Products:

PC6  
PC7

### Markets:

Avionics  
Homeland Security  
Industrial Automation  
Medical  
Petrochemical  
Printing

BEST AVAILABLE COPY

http://www.sbs.com/products/459

8/4/2005

EC6 - Single board computer based on Geode GX1 processor with PC104+ expansion slot

Semiconductor  
Transportation  
Wireless



Copyright © 2002-2005 SBS Technologies, Inc.

Privacy Policy : Terms & Conditions



A-vi

42390P5943C

## Hardware Requirements:

☒ Energy CIO/Information Technology Standards Program logo

## Hardware & System Requirements

for

### Microsoft Windows 2000 and Microsoft Office 2000

Microsoft Hardware Requirements (In Word)

Operating System (OS)	Hardware/System Requirements:
<b>Windows 2000 Professional</b>	<b>CPU</b>  133 MHz or higher Pentium-compatible CPU  300 MHz recommended  Supports single and dual CPU systems  NOTE: Check driver availability for peripheral devices.  <b>RAM</b>  64 MB (minimum) – 4 GB RAM (maximum)  128 recommended  <b>Disk Space:</b>  2 GB Hard disk  650 MB free installed on standalone PC  700 MB if installed over server  1G recommended
For more detailed information, visit the Microsoft web site:  <a href="http://www.microsoft.com/windows2000/upgrade/upgradereqs/default.asp">www.microsoft.com/windows2000/upgrade/upgradereqs/default.asp</a>	
<b>Applications</b>	<b>Hardware/System Requirements:</b>
<b>Office 2000 Premium</b> includes: Word, Excel, PowerPoint, Outlook, Access, FrontPage, Publisher, Small Business Tools, PhotoDraw	<b>For installing a typical configuration to the local PC</b>  <b>CPU:</b>  166 MHz Pentium-compatible CPU or higher  MS Windows 95 or later OS or

## Hardware Requirements:

	<p>MS NT Workstation OS ver. 4.0; Service pack 3 or later</p> <p>300 MHz Pentium-compatible CPU or higher recommended</p> <p><b>RAM:</b></p> <p>16 MB RAM for Windows 95/98 OS and 32 MB for Windows NT Workstations</p> <p>Additional 4 MB RAM for each application running simultaneously, except:</p> <p>8 MB required for Outlook, Access, FrontPage</p> <p>16 MB required for PhotoDraw</p> <p>Recommended total 256 MB to accommodate Office suite, other applications, toolbar, etc.</p> <p><b>Disk Space:</b></p> <p>252 MB for Word, Excel, PowerPoint, Outlook, Access, FrontPage</p> <p>174 MB Published, Small Business Tools</p> <p>100 MB PhotoDraw</p> <p>CD-ROM required for non-network installations</p> <p>Network Card required for network installations</p>
<p><b>Office 2000 Professional</b> includes: Word, Excel, Outlook, PowerPoint, Access, Publisher, Small Business Tools</p>	<p><b>For installing a typical configuration to the local PC</b></p> <p><b>CPU:</b></p> <p>75 MHz Pentium-compatible CPU or higher for</p> <p>MS Windows 95 or later OS or</p> <p>MS NT Workstation OS ver. 4.0; Service pack 3 or later</p> <p>300 MHz Pentium-compatible CPU or higher recommended</p> <p><b>RAM:</b></p> <p>16 MB RAM for Windows 95/98 OS and 32 MB for Windows NT Workstations</p> <p>Additional 4 MB RAM for each application running simultaneously, except:</p>

## Hardware Requirements:

Page 3 of 4

	<p>8 MB required for Outlook</p> <p>8 MB required for Access</p> <p>Recommended total 256 MB to accommodate Office suite, other applications, toolbar, etc.</p> <p><b>Disk Space:</b></p> <p>217 MB Word, Excel, PowerPoint, Outlook, Access</p> <p>174 MB Publisher, Small Business Tools</p> <p>CD-ROM required for non-network installations</p> <p>Network Card required for network installations</p>
<p><b>Office 2000 Standard</b></p> <p>Includes: Word, Excel, Outlook, PowerPoint</p>	<p><b>For installing a typical configuration to the local PC</b></p> <p><b>CPU:</b></p> <p>75 MHz Pentium-compatible CPU or higher for</p> <p>MS Windows 95 or later OS or</p> <p>MS NT Workstation OS ver. 4.0; Service pack 3 or later</p> <p>300 MHz Pentium-compatible CPU or higher recommended</p> <p><b>RAM:</b></p> <p>16 MB RAM for Windows 95/98 OS and 32 MB for Windows NT Workstations</p> <p>Additional 4 MB RAM for each application running simultaneously, except:</p> <p>8 MB required for Outlook</p> <p>Recommended total 256 MB to accommodate Office suite, other applications, toolbar, etc.</p> <p><b>Disk Space:</b></p> <p>189 MB Word, Excel, PowerPoint, Outlook</p> <p>CD-ROM required for non-network installations</p> <p>Network Card required for network installations</p>
<p><i>For more detailed information, visit the Microsoft web site:</i></p> <p><a href="http://www.microsoft.com/office/sysreq.htm">www.microsoft.com/office/sysreq.htm</a></p>	

Hardware Requirements:

Page 4 of 4

[Return to Previous Page](#)

---

[IT Reform](#) | [Standards Home](#) | [CIO Home](#) | [Energy.gov](#) | [Disclaimer](#)  
Comments or Questions regarding this site can be sent to the [Webmaster](#)  
Comments or Questions regarding the IT Standards Program should be sent to the  
IT Standards Manager, Carol Blackston [carol.blackston@hq.doe.gov](mailto:carol.blackston@hq.doe.gov)

A-vii

42390P5943C



### Product Information

## MOM 2005 System Requirements

Updated: August 25, 2004

Find the recommended hardware and software requirements for MOM 2005 and MOM 2005 Workgroup Edition. The page includes requirements for running a MOM server, database, consoles, and managed computers.

» » »

### On This Page

- ↓ [System Requirements for each MOM Server](#)
- ↓ [Database Server Requirements](#)
- ↓ [Administrator and Operator Console Requirements](#)
- ↓ [Managed Computer Requirements for each Computer](#)
- ↓ [MOM Reporting Server Requirements](#)
- ↓ [SQL Server 2000 Reporting Services Server Requirements](#)

### Related Links

- [How to Buy](#)
- [MOM 2005 Product Overview](#)
- [Compare MOM 2005 and Workgroup Edition](#)

## System Requirements for each MOM Server

Requirement	MOM 2005	MOM 2005 Workgroup Edition
<b>Processor</b>	PC with 550 MHz or higher Pentium-compatible	PC with 550 MHz or higher Pentium-compatible
<b>Operating system</b>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>• Microsoft Windows 2000 Server with the latest service pack</li> <li>• Microsoft Windows 2000 Advanced Server with the latest service pack</li> <li>• Microsoft Windows 2000 Datacenter Server with the latest service pack</li> </ul>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> </ul>
<b>Database software</b>	N/A	Any of the following:

## Microsoft Operations Manager 2005 System Requirements

		<ul style="list-style-type: none"> <li>• Microsoft SQL Server 2000 Desktop Engine (MSDE)</li> <li>• Microsoft SQL Server 2000 Standard Edition</li> <li>• Microsoft SQL Server 2000 Enterprise Edition</li> </ul>
<b>Memory</b>	512 megabytes (MB)	512 MB
<b>Hard disk</b>	5 GB	5 GB
<b>Hardware</b>	<ul style="list-style-type: none"> <li>• CD-ROM drive</li> <li>• Network adapter</li> <li>• Microsoft Mouse or compatible pointing device</li> </ul>	<ul style="list-style-type: none"> <li>• CD-ROM drive</li> <li>• Network adapter</li> <li>• Microsoft Mouse or compatible pointing device</li> </ul>

[Top of page](#)

## Database Server Requirements

Requirement	MOM 2005	MOM 2005 Workgroup Edition
<b>Processor</b>	PC with 550 MHz or higher Pentium-compatible	N/A
<b>Operating system</b>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>• Microsoft Windows 2000 Server with the latest service pack</li> <li>• Microsoft Windows 2000 Advanced Server with the latest service pack</li> <li>• Microsoft Windows 2000 Datacenter Server with the latest service pack</li> </ul>	N/A
<b>Memory</b>	512 MB	N/A
<b>Hard disk</b>	5 GB	N/A



## Microsoft Operations Manager 2005 System Requirements

<b>Database</b>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft SQL Server 2000 Standard Edition</li> <li>• Microsoft SQL Server 2000 Enterprise Edition</li> </ul>	N/A
<b>Hardware</b>	<ul style="list-style-type: none"> <li>• CD-ROM drive</li> <li>• Network adapter</li> <li>• Microsoft Mouse or compatible pointing device</li> </ul>	N/A

[↑ Top of page](#)

## Administrator and Operator Console Requirements

Requirement	MOM 2005	MOM 2005 Workgroup Edition
<b>Processor</b>	PC with 500 MHz or higher Pentium-compatible	PC with 500 MHz or higher Pentium-compatible
<b>Operating system</b>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft Windows XP Professional with the latest service pack</li> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>• Microsoft Windows 2000 Server with the latest service pack</li> <li>• Microsoft Windows 2000 Professional with the latest service pack</li> <li>• Microsoft Windows 2000 Advanced Server with the latest service pack</li> <li>• Microsoft Windows 2000 Datacenter Server with the latest service pack</li> </ul>	Any of the following: <ul style="list-style-type: none"> <li>• Microsoft Windows XP Professional with the latest service pack</li> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> </ul>

## Microsoft Operations Manager 2005 System Requirements

<b>Memory</b>	128 MB (minimum)	128 MB (minimum)
<b>Hard disk</b>	150 MB	150 MB
<b>Monitor resolution</b>	1024×768 or higher	1024×768 or higher
<b>Software</b>	Microsoft .NET Framework 1.1 or later	Microsoft .NET Framework 1.1 or later
<b>Hardware</b>	<ul style="list-style-type: none"> <li>• Network adapter</li> <li>• Microsoft Mouse or compatible pointing device</li> </ul>	<ul style="list-style-type: none"> <li>• Network adapter</li> <li>• Microsoft Mouse or compatible pointing device</li> </ul>

[↑ Top of page](#)

## Managed Computer Requirements for each Computer

<b>Requirement</b>	<b>MOM 2005</b>	<b>MOM 2005 Workgroup Edition</b>
<b>Processor</b>	PC with 200 MHz or higher Pentium-compatible	PC with 200 MHz or higher Pentium-compatible
<b>Operating system</b>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>• Microsoft Windows XP Professional with the latest service pack</li> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Web Edition with the latest service pack</li> <li>• Microsoft Windows Small Business Server 2003 with the latest service pack</li> <li>• Microsoft Windows 2000 Server with the latest service pack</li> <li>• Microsoft Windows 2000 Professional with the latest service pack</li> <li>• Microsoft Windows 2000 Advanced Server with the</li> </ul>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>• Microsoft Windows XP Professional with the latest service pack</li> <li>• Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>• Microsoft Windows Server 2003, Web Edition with the latest service pack</li> <li>• Microsoft Windows Small Business Server 2003 with the latest service pack</li> <li>• Microsoft Windows 2000 Server with the latest service pack</li> <li>• Microsoft Windows 2000 Professional with the latest service pack</li> <li>• Microsoft Windows 2000 Advanced Server with the</li> </ul>

## Microsoft Operations Manager 2005 System Requirements

Page 5 of 8

	<p>latest service pack</p> <ul style="list-style-type: none"> <li>● Microsoft Windows 2000 Datacenter Server with the latest service pack</li> <li>● Microsoft Windows NT 4.0 Server with the latest service pack (agent-less monitoring only)</li> <li>● Microsoft Windows NT 4.0 Server Enterprise Edition with the latest service pack (agent-less monitoring only)</li> <li>● Microsoft Windows NT 4.0 Server Terminal Server Edition with the latest service pack (agent-less monitoring only)</li> </ul>	<p>latest service pack (agent-less monitoring only)</p> <ul style="list-style-type: none"> <li>● Microsoft Windows 2000 Datacenter Server with the latest service pack (agent-less monitoring only)</li> <li>● Microsoft Windows NT 4.0 Server with the latest service pack (agent-less monitoring only)</li> <li>● Microsoft Windows NT 4.0 Server Enterprise Edition with the latest service pack (agent-less monitoring only)</li> <li>● Microsoft Windows NT 4.0 Server Terminal Server Edition with the latest service pack (agent-less monitoring only)</li> </ul>
<b>Memory</b>	128 MB (minimum)	128 MB (minimum)
<b>Hard disk</b>	100 MB	100 MB

[↑ Top of page](#)
**MOM Reporting Server Requirements**

<b>Requirement</b>	<b>MOM 2005</b>	<b>MOM 2005 Workgroup Edition</b>
<b>Processor</b>	PC with 550 MHz or higher Pentium-compatible	N/A
<b>Operating System</b>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>● Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>● Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>● Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>● Microsoft Windows 2000 Server with the latest service pack</li> <li>● Microsoft Windows 2000 Advanced Server with the latest service pack</li> </ul>	N/A

## Microsoft Operations Manager 2005 System Requirements

	<ul style="list-style-type: none"> <li>Microsoft Windows 2000 Datacenter Server with the latest service pack</li> </ul>	
<b>Database software</b>	Any of the following: <ul style="list-style-type: none"> <li>Microsoft SQL Server 2000 Standard with the Service Pack 3.0a or later</li> <li>Microsoft SQL Server 2000 Enterprise Edition with Service Pack 3.0a or later</li> </ul>	N/A
<b>Memory</b>	512 MB of RAM (1 GB or higher recommended)	N/A
<b>Hard disk</b>	200 GB of available hard disk space	N/A
<b>Hardware</b>	Any of the following: <ul style="list-style-type: none"> <li>CD-ROM drive or DVD-ROM drive</li> <li>Keyboard and mouse or compatible pointing device, or hardware that supports console redirection</li> <li>Network Adapter</li> </ul>	N/A

[↑ Top of page](#)

### SQL Server 2000 Reporting Services Server Requirements

**Note:** MOM 2005 Reporting utilizes SQL Server 2000 Reporting Services - you will need to install and configure SQL Server 2000 Reporting Services to view MOM 2005 Reports.

Requirement	MOM 2005	MOM 2005 Workgroup Edition
<b>Processor</b>	PC with 550 MHz or higher Pentium-compatible	N/A
<b>Operating System</b>	Any of the following: <ul style="list-style-type: none"> <li>Microsoft Windows Server 2003, Standard Edition with the latest service pack</li> <li>Microsoft Windows Server 2003, Enterprise Edition with the latest service pack</li> <li>Microsoft Windows Server 2003, Datacenter Edition with the latest service pack</li> <li>Microsoft Windows 2000</li> </ul>	N/A

## Microsoft Operations Manager 2005 System Requirements

	<p>Server with the latest service pack</p> <ul style="list-style-type: none"> <li>● Microsoft Windows 2000 Advanced Server with the latest service pack</li> <li>● Microsoft Windows 2000 Datacenter Server with the latest service pack</li> </ul>	
<b>Database software</b>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>● Microsoft SQL Server 2000 Standard with the Service Pack 3.0a or later</li> <li>● Microsoft SQL Server 2000 Enterprise Edition with Service Pack 3.0a or later</li> <li>● Microsoft SQL Server 2000 Reporting Services</li> </ul>	N/A
<b>Other Software</b>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>● Internet Information Services (IIS) Server 6.0 must be installed as part of the Windows Server installation</li> <li>● Microsoft SQL Server 2000 Reporting Services can render reports in HTML 3.2 and HTML 4.0. To view MOM reports you must have one of the following browsers:  Microsoft Internet Explorer 6.0 with Service Pack 1 Microsoft Internet Explorer 5.5 with Service Pack 2 Microsoft Internet Explorer 5.01 with Service Pack 2 Netscape 7.0 Netscape 4.78</li> <li>● Microsoft Visual Studio .NET 2003, or Integrated Developer Environment 2003 (if you want to customize or create reports)</li> </ul>	N/A
<b>Memory</b>	256 MB of RAM (1 GB or higher recommended)	N/A
<b>Hard disk</b>	10 GB of available hard disk space	N/A

## Microsoft Operations Manager 2005 System Requirements

Hardware	Any of the following: <ul style="list-style-type: none"><li>● CD-ROM drive or DVD-ROM drive</li><li>● Keyboard and mouse or compatible pointing device, or hardware that supports console redirection</li><li>● Network Adapter</li></ul>	N/A
----------	---	-----

[↑ Top of page](#)[Manage Your Profile](#)© 2005 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)**Microsoft**

A-viii

THIS PAGE BLANK (USPTO)

42390P5943C



# A42

Home: - Español - Français - Deutsch - Chinese  
Recent Posts - Books - FAQs

## Who's new

- thisisdeepan
- garygary40
- metrakos
- nicolas
- mms177

## News Links

eAsylum.net De Fr Es  
Bellacio Fr It  
Golem.de En Fr  
Heise.de En Fr  
HispaLinux En  
LinuxFr.org En De  
KnowProSE De Fr Es  
LinuxOnline.cn En  
LinuxOrg.ru En  
LinuxRu.net En  
Linux.TCPIP.cn En  
LinuxWorld.au De Fr  
LWN De Fr  
MozillaQuest De Fr  
SoftwareLivre.org En  
ZDNet De Fr  
ZDNet.cn En  
ZDNet.de En Fr  
ZDNet.fr En De

## Sponsors -

## OpenOffice 1.1 Competitive Guide SMB Segment

Submitted by Wonko The Sane on 26 March, 2004 - 09:05. Technical & Science

I came across OpenOffice 1.1 Competitive Guide SMB Segment through a post on the TTLUG mailing list, and decided to answer it fully in a FDL'd response because it will save quite a few people from typing everything.

### The Basics

In Microsoft's related document (PDF), the basic system requirements of OpenOffice are:

- \* Windows (98, NT, 2000, XP) - Pentium-compatible PC, 64 MB RAM, 130 MB HD; or
- \* Linux (x86, PowerPC) - 64 MB RAM and 170 MB HD
- \* Solaris (x66, SPARC) - 64 MB RAM and 240 MB HD; or
- \* MacOSX (beta); or
- \* FreeBSD

They did not, however, compare it to Office XP. We shall through Microsoft's own Office XP System requirements:

- \* Computer with Pentium 133 megahertz (MHz) or higher processor; Pentium III recommended
- \* # Windows 98, or Windows 98 Second Edition 24 MB of RAM plus an additional 8 MB of RAM for each Office program (such as Microsoft Word) running simultaneously
- \* Windows Me, or Microsoft Windows NT®
- \* 32 MB of RAM plus an additional 8 MB of RAM for each Office program (such as Word) running simultaneously

## Quotable

"Enfin, la thèse selon laquelle l'Open source menace l'industrie américaine de l'édition logicielle ne se vérifie pas au vu des résultats financiers de ses acteurs."  
— Olivier Le Quézourec

## User login

Username:

Password:



- Create new account
- Request new password

## Navigation

- ▢ recent posts



Ads by Goooooogle

#### Free Windows XP Pro

Get MS Windows XP Pro w/ SP 2 Free Guaranteed 100% Free - Act Now!  
www.InternetOpinionGrot

#### Repair Windows XP Errors

Fix all errors in minutes! Free download and scan. aff  
ErrorNuker.com

#### Registry Cleaner Download

Free Registry Scan, fix errors and Improve performance - 5 Star Rated.  
www.pctools.com

#### Free Virus Scan

Check for viruses, trojans & worms. Always up-to-date. Download now!  
www.Stop-Sign.com

#### eAsylum.net

\* Windows 2000 Professional  
64 MB of RAM plus an additional 8 MB of RAM for each Office program (such as Word) running simultaneously

\* Windows XP Professional, or Windows XP Home Edition 128 MB of RAM plus an additional 8 MB of RAM for each Office program (such as Word) running simultaneously

\* Hard disk space requirements will vary depending on configuration; custom installation choices may require more or less. Listed below are the minimum hard disk requirements for Office XP suites:

\* Office XP Standard

210 MB of available hard disk space

\* Office XP Professional and Professional Special Edition<sup>2</sup>

245 MB of available hard disk space

An additional 115 MB is required on the hard disk where the operating system is installed. Users without Windows XP, Windows 2000, Windows Me, or Office 2000 Service Release 1 (SR-1) require an extra 50 MB of hard disk space for System Files Update.

\* Windows 98, Windows 98 Second Edition, Windows Millennium Edition (Windows Me), Windows NT 4.0 with Service Pack 6 (SP6) or later,<sup>3</sup> Windows 2000, or Windows XP or later.

\* CD-ROM drive

\* Super VGA (800 x 600) or higher-resolution monitor with 256 colors

\* Microsoft Mouse, Microsoft IntelliMouse®, or compatible pointing device

Please do not forget the key phrase in these Office XP requirements: "an additional 8 MB of RAM for each Office program (such as Microsoft Word) running simultaneously". That said, **OpenOffice** more than holds its own, and does so in less disk space on more operating systems.

Being functional on more operating systems guarantees more cross compatibility between platforms, which allows users to change their operating systems, if they so decide, with a lower migration cost. So **OpenOffice's** customizability could actually decrease costs in the future; it is not reliant on one operating system.

The Minimum Office XP requirements state 'Pentium 133 MHz machine'. In

## Blogs

- Wonko's Presidential Debate
- Cyber-Hitchhiker: New Editor Smell
- Hitchhiker's Diary - Sublimation
- Hitchhiker's Diary - Who Hates Freedom?
- Hitchhiker's Diary - Even More Reasons to Become a Hitchhiker
- Hitchhiker's Diary - Objective: China
- Hitchhiker's Diary - From Dubai
- Hitchhiker's Diary: Philosophical Discussion with Another Grenadan
- Hitchhiker's Diary - Latin American Traffic Rules
- Hitchhiker's Diary - Life as Entertainment

more

## Hitchhikers

There are currently 0 users and 24 guests online.

## Top nodes

Today's top:

- A42 Front Page
- Links for WRT54G Linux Hacks (Update)
- replica whatcha handbags jewelry==
- MMJB

translation, this would probably be a machine used by a Windows 98SE user, which would require them to have 24 - 56 Megabytes of RAM, 375 Megabytes of Hard disk space, etc.

The XP requirements for Office XP are much more interesting. 128-168 megabytes of memory and 325 megabytes of hard drive space.

**Let's compare again with the OpenOffice requirements for X<sup>2</sup> and 98SE:**

Windows (98, NT, 2000, XP) – Pentium-compatible PC, 64 MB RAM, 130 MB HD\*.

**Clear winner: OpenOffice.**

**Now we shall look at their 'Value Proposition And Response'.**

## Value Proposition And Response

Microsoft's document stresses that the licensing costs are not representative of the total costs of ownership, and this is a valid point. But let's compare, point by point:

**\* Installation and deployment:** OpenOffice can be installed at no cost, and deployed easily. Microsoft Office XP, however, requires licensing costs and requires more hardware to run on (see above). It also requires that you run an operating system which must be licensed at cost. An international comparison of **cost per license of operating system and GPP** is revealing in this regard.

**\* Data Migration and Testing:** In migrating Microsoft Office documents to OpenOffice, some advanced formatting may be lost - and this is a problem, but it is unreasonable to demand this because of the fact that Microsoft does not make it's data formats public.

They make special note on the cost of migrating a Microsoft Access database to OpenOffice, but fail to mention the costs associated with upgrading a Microsoft Access database even with their own software. **Free Software and Open Source** databases are typically available at no cost, so the money would be spent on the actual 'liberation' of the data. Microsoft will require you to purchase licensing for SQL Server, and

- [illegible]

**more**

**KnowProSE.com**

- **Funeral For My Father on August 4th, 2005.**
- **My Father, Mahindranath Taran Pande Rampersad Has Passed Away.**
- **Reprieve From Exposure to Global Insanity**
- **J. Krishnamurti**
- **IBM maintenance manual (1925)**
- **It's Been A Long Week**
- **Intellectual Property, The Caribbean, and Everything**
- **Constitution of Guyana**
- **Thoughts on the Caribbean Blog List**
- **Caribbean Blog List**
- **Hospital Food Can Kill**

**All time top:**

- **A42 Front Page**
- **Playboy + McDonalds = SuperSize Cholesterol SexBombs?**
- **Mandrake Issued Cease and Desist, Must Change Name**
- **Links for WRT54G Linux Hacks (Update)**

**Last:**

- **A42 Front Page**
- **[Germany] Government Releases Open Source Migration Manual**
- **Playboy + McDonald's = SuperSize Cholesterol SexBombs?**

businesses will still have to pay for the migration of the data.

- **You: Trinidad**
- **Guayanesa Women Lured Into Prostitution By Trinidad Company?**
- **The Perfect Medical System**
- **The Trinidad Train System**
- **Categorizations and The Internet; Names as Fences - and more Questions than Answers**

more

## Syndicate

XML

\* **Email client:** Microsoft notes that **OpenOffice** lacks an email client.

This, however, would take us to **Mozilla**, which is a standalone web browser with more features than Internet Explorer (such as tabbed browsing), and is much more secure than Microsoft Outlook as a default.

\* **Collaboration:** Microsoft makes it a point to discuss that collaboration is required. Yet **OpenOffice** runs on all major operating systems, and Microsoft Office does not. This certainly becomes an issue of collaboration.

They also mention that there is a need to assure mission critical data is impervious to virus attack - and given the latest viruses, this does not bode well for them as all major attacks have taken advantages of flaws in Microsoft Operating Systems and even their Office software. This can lead down the path to security itself, in which ubiquity of Microsoft products probably has an effect.

\* **Support:** Microsoft says that there is no dedicated team for the OpenOffice suite. What Microsoft fails to realize is that the 'dedicated team' are mainly the users; **OpenOffice** has a community whereas Microsoft users have support groups.

\* **Limited Compatibility:** Microsoft properly asserts that OpenOffice is not 100% compatible with their product. Microsoft, however, has apparently decided not to support the **OpenOffice** formats either, for which they have no excuse: the standards for OpenOffice documents are publicly available, whereas Microsoft makes it a habit to sue people for reverse engineering

their own formats. **Richard Stallman wrote about this in 2002.**

#### Total Value Of OpenOffice

- (1) **Ease of Use:** While computer users throughout the world (including this author) have become familiar with Microsoft's Office suites over the years, **OpenOffice** is not difficult to learn just by simply *using* it. It's been long kept a secret, but no training in basic use of Office suites is needed; advanced use of the Office suites may constitute need for training regardless of which suite it is.
- (2) **Tailored Solutions:** **OpenOffice** has the benefit of permitting more customized applications to interact with it due to the Freedom associated with the source code, which means it will allow more people to develop custom applications which interact with it. Microsoft products require more Microsoft products to interact with them, they come at a cost and *limit* what a developer can do since the source code is not available.
- (3) **Better and Faster Work:** Such comparisons are inherently flawed, since they would have to have the same users doing the same work on different Office suites. Let's face it: Users just want to do what they have to with their software. In this regard, OpenOffice facilitates this just as Microsoft Office does, but has the benefit of having the source code available for allowing more applications to interact with it. This means more potential productivity when dealing with the business logic of a SMB.
- (4) **Seamless Data Exchange:** Microsoft claims seamless data exchange within Microsoft Office - but it's only between people using Microsoft products. OpenOffice allows people who use a variety of operating systems and data formats to interact with each other. Microsoft Office does not.
- (5) **Easier Deployment and Maintenance:** Installation for either package is very simple. OpenOffice does have a clear benefit here: Service packs are not something one has to constantly look for (at this time). Further, simply installing the latest version of OpenOffice over a later version takes less overall time than constantly updating via patches and service packs.
- (6) **Security:** Microsoft is brave to bring viruses into it's marketing strategy when it has been one of Microsoft's greatest problems, despite all the nice things their Marketing brochures have to say about how secure it is. Where the rubber meets the road, Microsoft Office loses.

(7) Investment You Can Trust: Using **OpenOffice** is an investment of your time, your energy and your future of being able to interoperate with people around the world, without worrying about what operating system that they use. Microsoft Office is an investment in Microsoft's time, energy and future.

#### Final Words

Microsoft used to have an advertisement asking where you wished to go today; this is more true of **OpenOffice** since it allows you more control of your data through vendors and even inhouse staff who can help with it. Microsoft is dictating a future; this is why they do not allow Open Standards.

This is also why Microsoft spends so much time in courts around the world.

*Copyright (c) 2004, Taran Rampersad.*

*Permission is granted to copy, distribute and/or modify this document under the terms of the **GNU Free Documentation License, Version 1.2** or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License" on the **GNU** website.*

This article was originally posted [here](#).

» [login or register](#) to post comments | 1727 reads

Unless specified elsewhere, copyright reserved by SSC, Inc.

<http://www.a42.com/node/view/142>

8/4/2005

A-ix

THIS PAGE BLANK (USPTO)

42390P5943C



All Topics, MFC / C++ >> System >> General

## Get the Processor Speed in two simple ways

By Thomas Latuske

Get the frequency of the processor either from the registry, or calculate it.

Search:  [Articles](#)

### Toolbox

Broken Article?

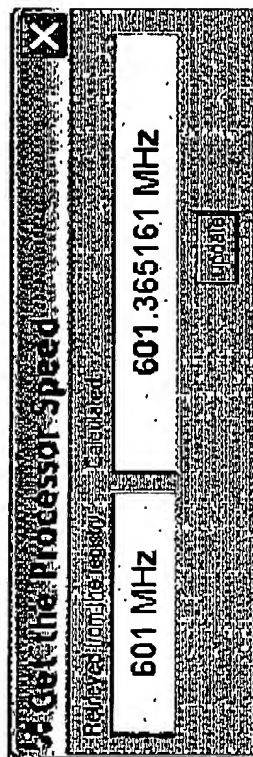
VS.NET 2003 for \$899

Ultimate Toolbox \$499

Print version

Send to a friend

Download demo project - 20.8 Kb



### Introduction

I'll show you two ways to retrieve the processor-speed (**frequency in MHz**). With two simple functions, one to retrieve the frequency from the registry of your Windows operating system, and one to calculate it with the clock cycles and a high resolution counter. If you want to use the function to calculate the speed (**frequency**), you have to use it with a Pentium instruction set compatible processor (look at the lines below).

**rfmobile** wrote in a message:

*You don't need to change the RDTSC definition for non-Intel processors. The code works as-is on my AMD mobile Athlon. Should work on any Pentium instruction set compatible processor but not for 486 or 386.*

View our advertisers

Advertise with us



C++ (VC6)  
Windows (NT4, WinXP)  
Win32, VS, MFC  
Dev

Posted 8 Jun 2004

Updated 13 Jun 2004

Articles by this author

17,648 views

[Help!](#) [Article Help!](#) [Articles](#) [Message Boards](#) [StoreFront](#)

28 members have rated this article. Result:

Popularity: 6.73. Rating: 4.65 out of 5.

Sign in / Sign up

Email

Password

☒ Remember me

Lost your Password?





## Routine to calculate the processor frequency in MHz:

Retrieve the frequency in MHz as a floating-point number. I use some well documented (at least for me ;-)) assembler here:

```
float CGetTheProcessorSpeedDlg::ProcSpeedCalc()
{
    /*
    Rdtsc:
    It's the Pentium instruction "Read Time Stamp Counter". It measures the
    number of clock cycles that have passed since the processor was reset, as a
    64-bit number. That's what the <CODE>_emit lines do.*/
    #define Rdtsc __asm _emit 0x0f __asm _emit 0x31

    // variables for the clock-cycles:
    __int64 cyclesStart = 0, cyclesStop = 0;
    // variables for the High-Res Performance Counter:
    unsigned __int64 nCtr = 0, nFreq = 0, nCtrStop = 0;

    // retrieve performance-counter frequency per second:
    if (!QueryPerformanceFrequency((LARGE_INTEGER *) &nFreq)) return 0;

    // retrieve the current value of the performance counter:
    QueryPerformanceCounter((LARGE_INTEGER *) &nCtrStop);

    // add the frequency to the counter-value:
    nCtrStop += nFreq;

    _asm {
        /// retrieve the clock-cycles for the start value:
        Rdtsc
        mov DWORD PTR cyclesStart, eax
        mov DWORD PTR [cyclesStart + 4], edx
    }

    do {
        /// retrieve the value of the performance counter
        /// until 1 sec has gone by:
        QueryPerformanceCounter((LARGE_INTEGER *) &nCtr);
    } while (nCtr < nCtrStop);

    _asm {
        /// retrieve again the clock-cycles after 1 sec. has gone by:
        Rdtsc
    }
```

```

mov DWORD PTR cyclesStop, eax
mov DWORD PTR [cyclesStop + 4], edx
}

// stop-start is speed in Hz divided by 1,000,000 is speed in MHz
return ((float)cyclesStop-(float)cyclesStart) / 1000000;
}

```

## Credits

- I got the assembler some time ago from an assembler newsgroup
- ...and credits to all programmers out there who share their knowledge!

## About Thomas Latuske



My name is Thomas, I'm born on January the 11th in 1970, right now I'm working in the Quality department of a big Pipe mill as a Technician.

My hobbies are my girl friend, my car, RC-Planes and Computers. I begun with VC++ some time ago and now Programming is like a drug to me (I'm still a beginner). I want to learn it all in a blink of an eye ☺ but I know that this is not possible. It's real fun for me and I do small Programs for my own use.

O.K. enough written..... I need my Time to debug everything that crosses my way! ☺

Click here to view **Thomas Latuske's** online profile.

## Other popular articles:

- Driver Development Part 1: Introduction to Drivers  
This article will go into the basics of creating a simple driver.
- Driver Development Part 2: Introduction to Implementing IOCTLs  
This article will go deeper into the basics of creating a simple driver.
- A simple demo for WDM Driver development

WDM Driver programming Introduction with three Pseudo Drivers.

• API hooking revealed

The article demonstrates how to build a user mode Win32 API spying system

Ads by Google

**AMD64 Processors**

Get Smarter, Faster Computing Performance w/AMD64. Learn More!

[www.monarchcomputer.com](http://www.monarchcomputer.com)

**Dual Pentium Motherboards**

Pentium Motherboards On Sale. Large Selection, Super Low Prices!

[www.tigardirect.com](http://www.tigardirect.com)

**AMD64 Processors**

Get Smarter, Faster Computing Performance w/AMD64. Learn More!

[www.boldata.com](http://www.boldata.com)

**AMD64 Processors**

Speed-Power-Performance for 32- & 64- Bit Computing. Learn More Now!

[www.pub.supercom.ca](http://www.pub.supercom.ca)

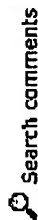
[Top]

Sign in to vote for this article: Poor ○ ○ ○ ○ ○ Excellent

**Visual Studio .NET IDE for Linux!**

Premium Sponsor

**Spell Check Your Code in the .NET IDE**  
Windows Forms & User Controls  
ASPX, ASCX, HTML, RESX files  
**SoftWidgets Spell Code**



Noise tolerance

Medium

Search comments

View Message View

Per page

25

New thread.

Msgs 1 to 15 of 15 (Total: 15) (Refresh)

First Prev Next

Subject

Author

Date

Why waiting that second?

Anonymous

5:58 29 Jul '04

Accuracy not guaranteed

DavidCRow

14:05 17 Jun '04

The code I'm posting here has always worked for me

Olan Patrick Barnes

15:14 14 Jun '04

Re: The code I'm posting here has always worked for me

Thomas Latuske

16:39 14 Jun '04

Get Processor Speed should be Get Processor Frequency

technomanceraus

1:30 9 Jun '04

Re: Get Processor Speed should be Get Processor Frequency

RichardGrimmer

10:18 9 Jun '04

Re: Get Processor Speed should be Get Processor Frequency

Anonymous

23:21 13 Jun '04

[http://www.codeproject.com/system/Processor\\_Speed.asp](http://www.codeproject.com/system/Processor_Speed.asp)

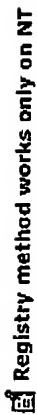
8/4/2005



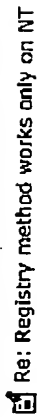
Re: Get Processor Speed should be Get Processor Frequency



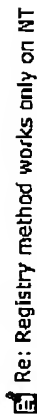
Re: RDTSC



Registry method works only on NT



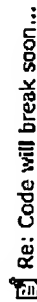
Re: Registry method works only on NT



Re: Registry method works only on NT



Code will break soon...



Re: Code will break soon...

Last Visit: 21:23 Thursday 4th August, 2005

All Topics, MFC / C++ >> System >> General  
Updated: 13 Jun 2004

The Ultimate Toolbox • MSDN Communities | ASP Alliance • Developer Fusion • Developersdex • DevGuru • Programmers Heaven • Planet Source Code • Tek-Tips  
Forums •

Article content copyright Thomas Latuske, 2004  
everything else Copyright © CodeProject, 1999-2005.  
Advertise on The Code Project | Privacy

First Prev Next

	Vince C.	4:22 15 Jun '04	
	rfmobile	21:07 8 Jun '04	
	Thomas Latuske	12:04 13 Jun '04	
	dacrls	15:39 8 Jun '04	
	Rick York	16:04 8 Jun '04	
	Thomas Latuske	16:43 8 Jun '04	
	DIWa	14:08 8 Jun '04	
	Thomas Latuske	15:04 8 Jun '04	

A-x

42390P5943C

THIS PAGE BLANK (USPTO)



US005859789A

## United States Patent [19]

**Sidwell**

[11] **Patent Number:** **5,859,789**

[45] **Date of Patent:** Jan. 12, 1999

154 ARITHMETIC UNIT

5,081,573	1/1992	Hall	395/800.02
5,446,651	8/1995	Moyse	364/760

5,446,651	8/1995	Moyse	364/760
-----------	--------	-------	---------

[75] Inventor: **Nathan M. Sidwell, St. Werburghs,  
United Kingdom**

## OTHER PUBLICATIONS

[73] Assignee: SGS-Thomson Microelectronics Limited, Almondsbury, United Kingdom

Standard Search Report issued by the European Patent Office and dated Feb. 13, 1996.

[21] Appl. No.: 677,837

Proceedings Of the 7th IEEE Symposium on Computer Arithmetic, 4-6 Jun. 1985, Urbana, IL, USA, 1985 IEEE Computer Society Press, Los Alamitos CA US, pp. 38-43. Electronic Design, vol. 32, No. 10, May 1984 Hasbrouck Heights, New Jersey US, pp. 191-206, W. Meshach "Data-Flow IC Makes Short Work Of Tough Processing Chores".

[22] Filed: Jul 10, 1996

*Primary Examiner—Eric Coleman*

[30] Foreign Application Priority Data

Attorney, Agent, or Firm—Wolf, Greenfield & Sacks, P.C.

Jul. 18, 1995 [GB] United Kingdom ..... 9514684

[51] Int. Cl.<sup>6</sup> ..... G06F 9/302

[52] U.S. CI. .... 364/750.5; 364/736.02

[58] **Field of Search** ..... 395/800.02, 841,  
395/376; 364/748.09, 748.2, 754.01, 754.02,  
758, 768, 765, 750.5, 748.07, 736.02, 806

[57] ABSTRACT

There is disclosed an arithmetic unit which allows a combined multiply-add operation to be carried out in response to execution of a single computer instruction. This is particularly useful in a packed arithmetic environment, when a operand comprises a plurality of packed objects and the intention is to carry out the same arithmetic operation on respective pairs of objects in different operands. There is also provided a computer and a method of operating a computer to effect the combined multiply-add operation.

[56] **References Cited**

## U.S. PATENT DOCUMENTS

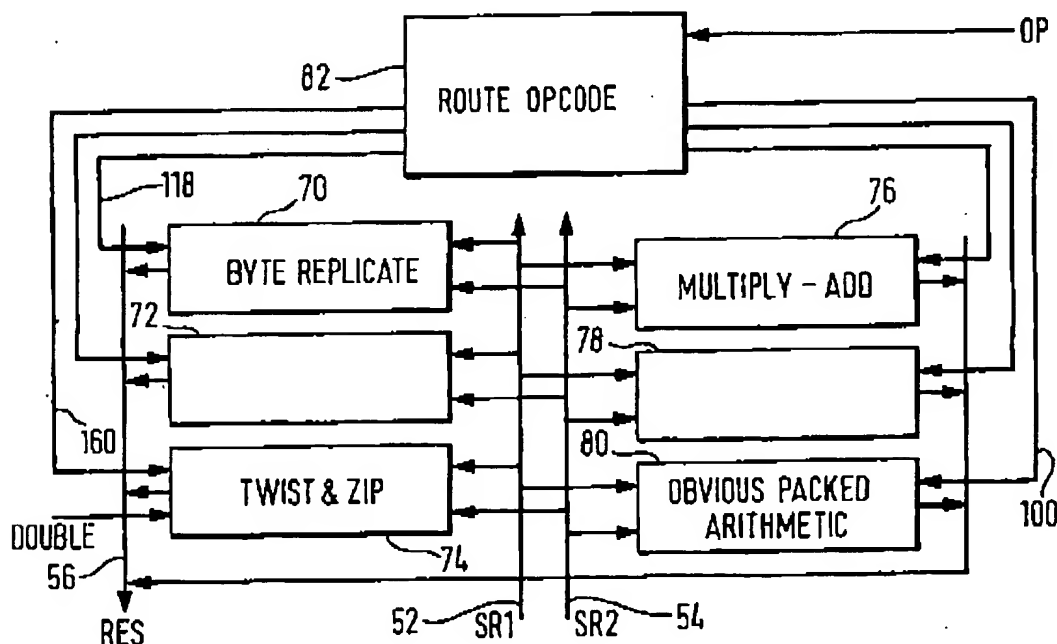
4.617.625 10/1986 Nagashima ..... 395/800.04

4,761,754	3/1988	Kinoshiya	364/736
-----------	--------	-----------	---------

4,771,379	9/1988	Hidachi et al.	364/200
-----------	--------	----------------	---------

4,888,682 12/1989 Ngai ..... 364/730

**13 Claims, 4 Drawing Sheets**

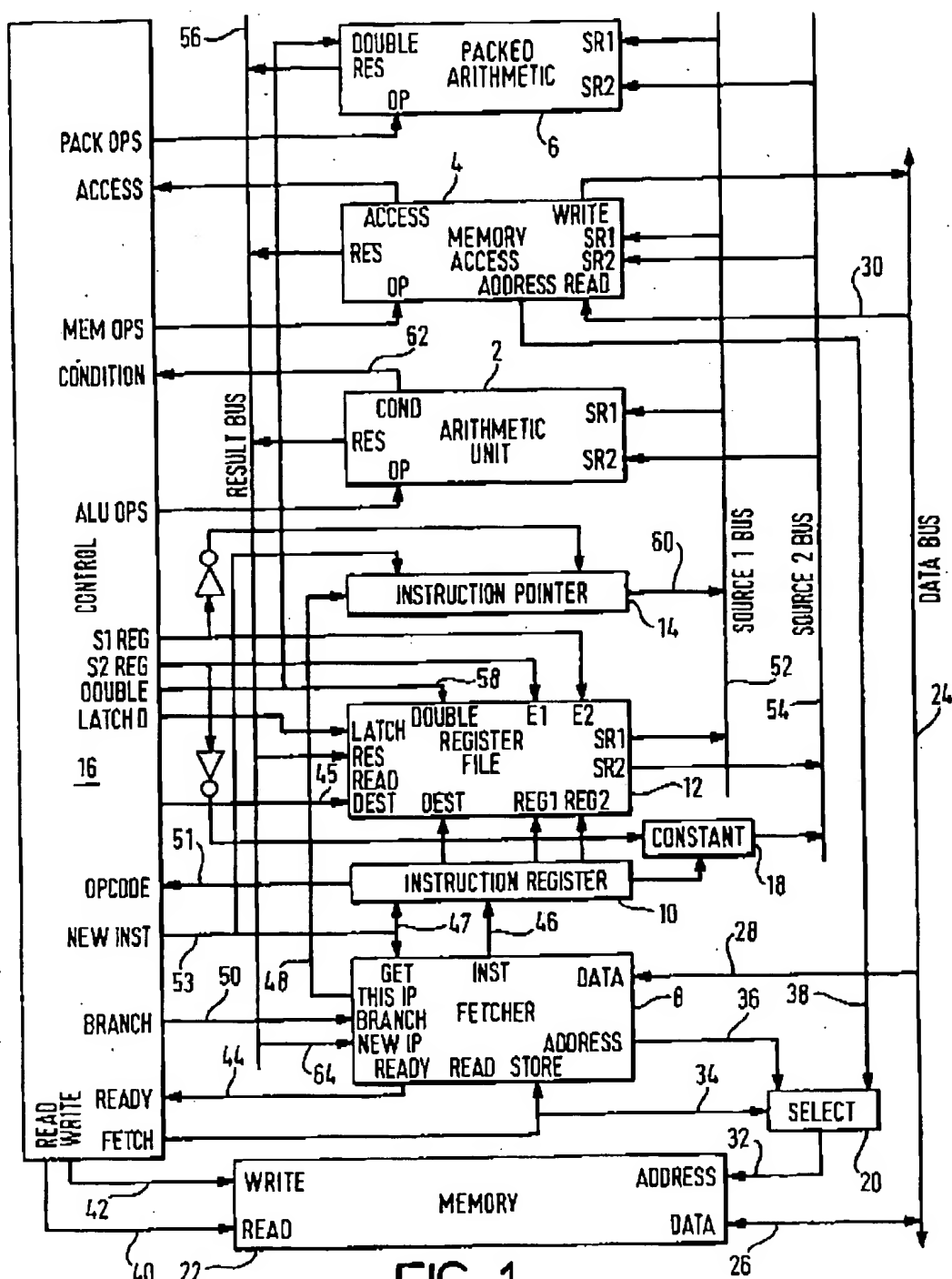


U.S. Patent

Jan. 12, 1999

Sheet 1 of 4

5,859,789



U.S. Patent

Jan. 12, 1999

Sheet 2 of 4

5,859,789

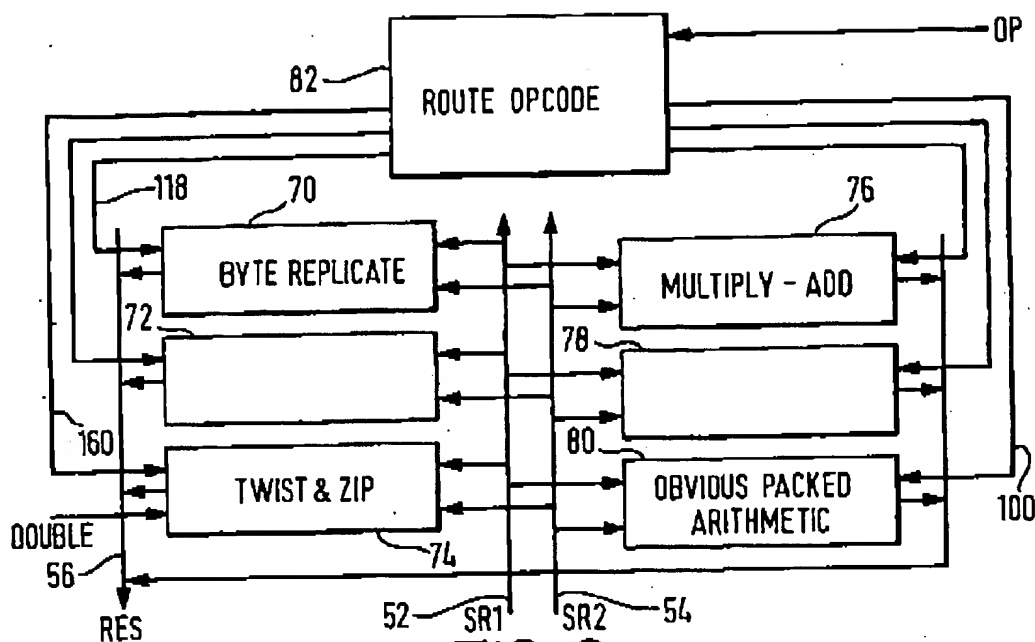


FIG. 2  
PACKED UNIT

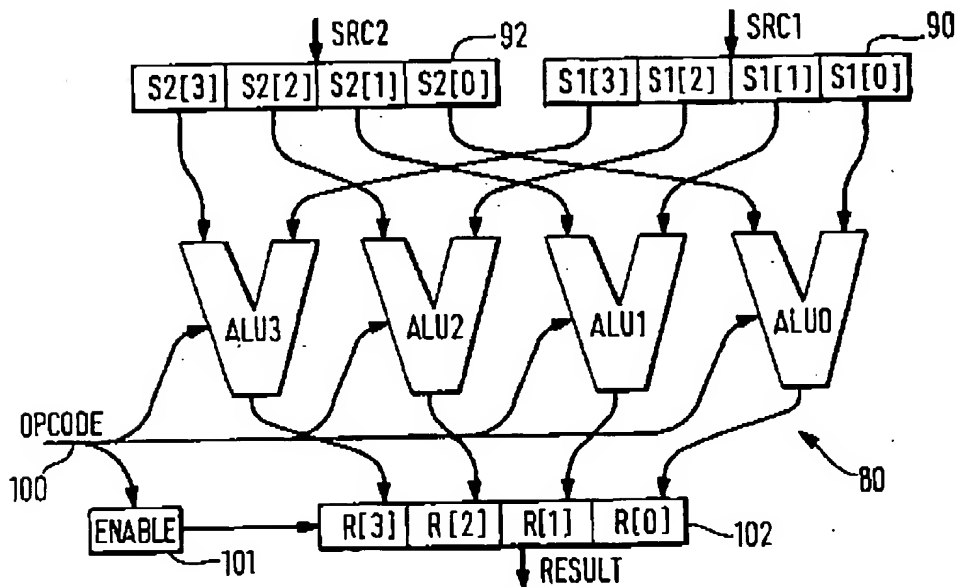


FIG. 4  
OBVIOUS PACKED ARITHMETIC


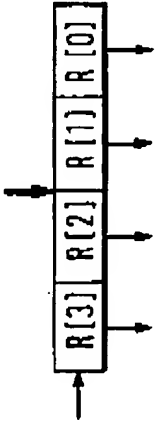
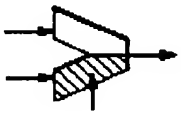



U.S. Patent

Jan. 12, 1999

Sheet 3 of 4

5,859,789

SYMBOL	OPERATION
	<p>ARITHMETIC LOGIC UNIT. COMBINES THE TWO SOURCE VALUES IN SOME MANNER TO PRODUCE A RESULT.</p>
	<p>BUFFER. THIS PARTICULAR ONE HAS AN INPUT WHICH IS UNSEPARATED AND FOUR SEPARATED OUTPUTS, EACH TAKING ONE QUARTER OF THE INPUT SIGNALS, IT ALSO HAS AN OUTPUT ENABLE INPUT.</p>
	<p>MULTIPLEXER. THE OUTPUT CONSISTS OF ONE OF THE TWO INPUT SIGNALS. WHEN THE CONTROL SIGNAL IS NOT ASSERTED, THE UNSHADED INPUT SIGNAL IS OUTPUT, WHEN THE CONTROL SIGNAL IS ASSERTED, THE SHADED INPUT SIGNAL IS OUTPUT.</p>
	<p>CHANGEOVER SWITCH. EACH OF THE TWO OUTPUTS CONSISTS OF ONE OF THE INPUT SIGNALS. WHEN THE CONTROL SIGNAL IS NOT ASSERTED, THE UNSHADED INPUT GOES TO THE UNSHADED OUTPUT AND THE SHADED INPUT GOES TO THE SHADED OUTPUT. WHEN THE CONTROL SIGNAL IS ASSERTED, THE OUTPUTS SWAP OVER.</p>

**FIG. 3**  
SYMBOLS

U.S. Patent

Jan. 12, 1999

Sheet 4 of 4

5,859,789

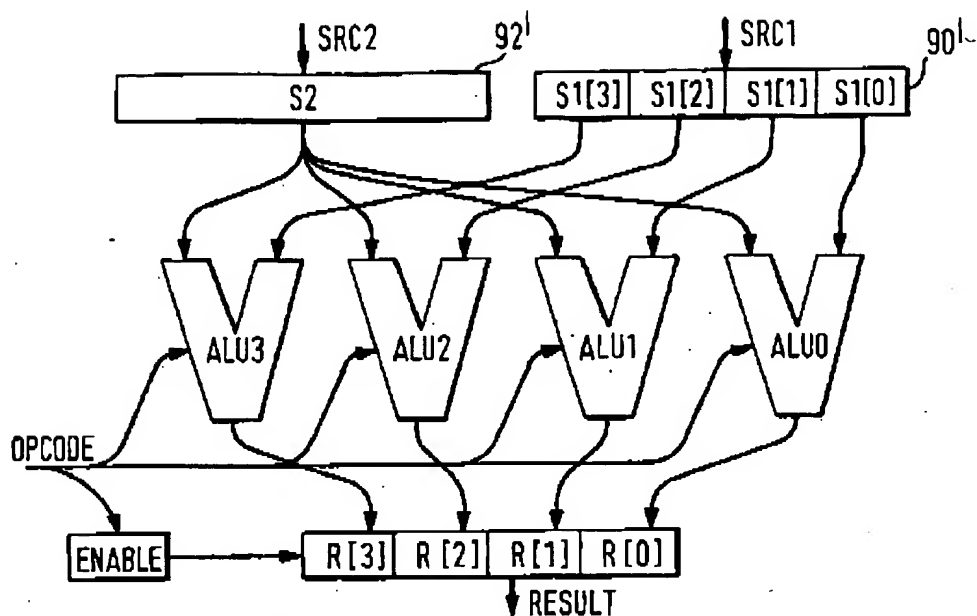


FIG. 5

OBVIOUS PACKED ARITHMETIC WITH UNPACKED OPERAND

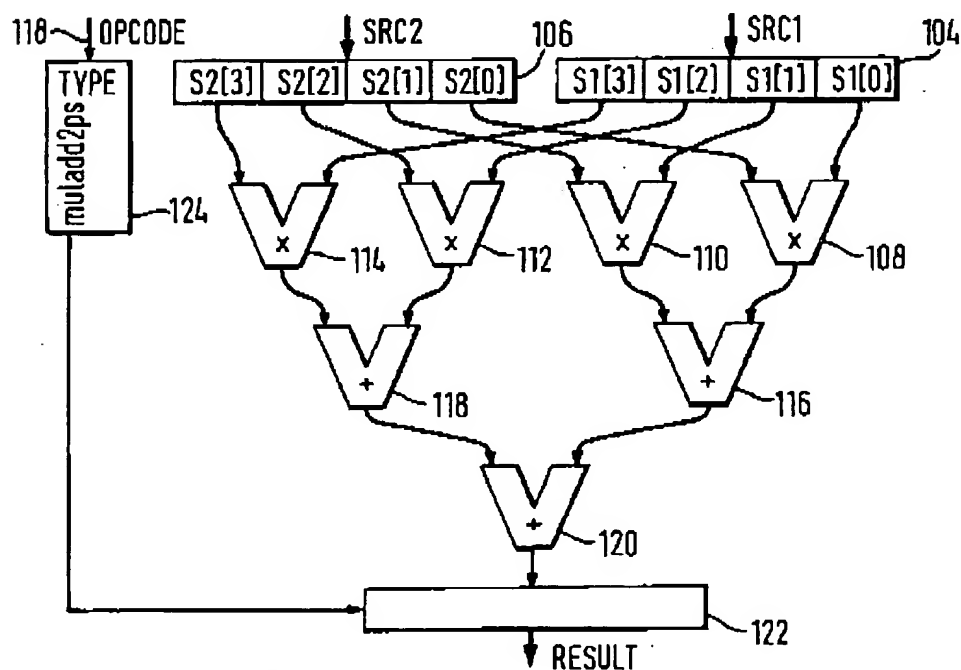


FIG. 6 MULTIPY ADD

5,859,789

1

## ARITHMETIC UNIT

## FIELD OF THE INVENTION

This invention relates to an arithmetic unit for use in a computer system.

## BACKGROUND TO THE INVENTION

Arithmetic units are those which carry out an arithmetic operation in response to execution of an arithmetic instruction. Such instructions include an add instruction, a multiply instruction, a divide instruction and a subtract instruction. It is common to have in a computer system a so-called ALU (arithmetic logic unit) which is capable of implementing any one of these arithmetic instructions.

There are frequent occasions when it is required to multiply together pairs of objects and to add together the resulting products. This is presently done by effecting a multiplication operation on each pair of objects, storing the results of the multiplication operations in a register file and subsequently executing an addition instruction which recalls the earlier generated results from the register file and adds them together, finally loading the result back to the register file. One problem with this arrangement is that the length of the words resulting from the multiplication operations are much longer than the original operands. For example, the multiplication together of two 16 bit objects will result in a 32 bit word. It is therefore necessary to have available register capacity to store these results. One way round this problem which is currently used is to introduce rounding to reduce the word lengths prior to storage. This however can introduce rounding errors and can result in the multiplication not being carried out to adequate precision.

Another problem is the requirement to execute two instructions, a multiplication instruction followed by an add instruction. Not only do these instructions take up space in the instruction sequence stored in memory but they take time to execute.

## SUMMARY OF THE INVENTION

According to one aspect of the present invention there is provided an arithmetic unit for executing an instruction to multiply together  $n$  pairs of objects and to add together the resulting products, said objects being represented by sub-strings of respective first and second data strings, the arithmetic unit comprising:

input buffer means for holding said first and second data strings;

a plurality ( $n$ ) of multiplication circuits for simultaneously multiplying together respective pairs of objects, each multiplication unit having a pair of inputs for receiving respective objects defined by sub-strings of each of the first and second data strings and providing an output;

addition circuitry connected to receive the outputs of the multiplication circuits and operable to add together the resulting products of multiplication of the respective pairs of objects to generate a result; and

output buffer means for holding the result.

This allows a combined multiply-add operation to be carried out in response to execution of a single instruction. It also has the advantage that the length of the result will always be less than the length of one of the data strings. Therefore, it can be ensured that the length of the result will not exceed the available capacity of the register for storing the result. This is particularly useful in a packed arithmetic environment, where an operand comprises a plurality of

2

packed objects and the intention is to carry out the same arithmetic operation on respective pairs of objects in different operands. The first and second data strings can constitute a single operand or can be separate operands.

In the preferred embodiment, the addition circuitry comprises a first set of adder circuits each arranged to add together the outputs of a distinct two of said  $n$  multiplication circuits, and a further adder circuit arranged to add together outputs of each of the first set of adder circuits to provide said result.

In the described embodiment, the input buffer means comprises first and second input buffers each arranged to hold a respective one of the first and second data strings. However, only one input buffer is needed in the case where pairs of objects within an operand are to be multiplied together. It will readily be apparent that it is an advantage of the invention that the output buffer means can have a capacity which is less than the input buffer means.

The invention also provides a computer comprising a processor, memory and data storage circuitry for holding data strings each comprising  $n$  sub-strings representing respective objects, wherein said processor comprises an arithmetic unit as defined above and wherein there is stored in said memory a sequence of instructions comprising at least an instruction to multiply together  $n$  pairs of objects and to add together the resulting products, said instruction being executed by the arithmetic unit.

The data storage circuitry can comprise a plurality of register stores each having a predetermined bit capacity matching the length of each of the data strings.

One particularly useful application of the combined multiply-add instruction described herein is to multiply a vector by a matrix. Thus, the invention further provides a method of operating a computer to multiply a vector by a matrix wherein the vector is represented by a vector data string comprising a plurality of substrings each defining vector elements and wherein the matrix is represented by a set of matrix data strings each comprising a plurality of sub-strings defining matrix elements, the method comprising selecting each of said matrix data strings in turn and executing for each selected data string a single instruction which:

loads the selected data string into an input buffer means of an arithmetic unit;

loads said vector data string into said input buffer means;

simultaneously multiplies respective pairs of vector elements and matrix elements of said data strings to generate respective products;

adds together said products; and

generates a result.

For a better understanding of the present invention and to show how the same may be carried into effect, reference will now be made by way of example to the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a processor and memory of a computer;

FIG. 2 is a block diagram of a packed arithmetic unit;

FIG. 3 shows the meaning of symbols used in the figures;

FIG. 4 is a block diagram of an obvious packed arithmetic unit operating on two packed source operands;

FIG. 5 is a block diagram of an obvious packed arithmetic unit which operates on a packed source operand and an unpacked source operand; and

FIG. 6 shows a multiply-add unit.

5,859,789

3

## DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 shows a processor in accordance with one embodiment of the present invention. The processor has three execution units including a conventional arithmetic unit 2 and a memory access unit 4. In addition there is a packed arithmetic unit 6. The processor also includes an instruction fetcher 8, an instruction register 10, a register file 12 and an instruction pointer 14 all of which operate under the control of a control unit 16 of the processor. The register file comprises a set of registers each having a predetermined bit capacity and each being addressable with a single address. It is not possible to address individual locations within a register. When a register is accessed, the entire contents of the register are concerned. The processor further includes a constant unit 18 and a select unit 20. The constant unit 18 and select unit 20 are also operated under the control of the control unit 16. The processor operates in conjunction with a memory 22 which holds instructions and data values for effecting operations of the processor. Data values and instructions are supplied to and from the memory 22 via a data bus 24. The data bus 24 supplied data values to and from the memory 22 via a memory data input 26. The data bus 24 also supplies data to the instruction fetcher 8 via a fetcher data input 28 and to the memory access unit 4 via a memory access read input 30. The memory is addressed via the select unit 20 on address input 32. The select unit 20 is controlled via a fetch signal 34 from the control unit 16 to select an address 36 from the fetcher 8 or an address 38 from the memory access unit 4. Read and write control lines 40, 42 from the control unit 16 control read and write instructions to and from the memory 22. The instruction fetcher 8 fetches instructions from the memory 22 under the control of the control unit 16 as follows. An address 36 from which instructions are to be read is provided to the memory 22 via the select unit 20. These instructions are provided via the data bus 24 to the fetcher data input 28. When the instruction fetcher has fetched its next instruction, or in any event has a next instruction ready, it issues a Ready signal on line 44 to the control unit 16. The instruction which is to be executed is supplied to the instruction register 10 along instruction line Inst 46 and held there during its execution. The instruction pointer 14 holds the address of the instruction being executed supplied to it from the fetcher 8 via instruction pointer line 48. A Get signal 47 responsive to a New Inst signal 53 from the control unit 16 causes the instruction register 10 to store the next instruction on Inst line 46 and causes the fetcher 8 to prepare the next instruction. The New Inst signal 53 also causes the instruction pointer 14 to store the address of the next instruction. A branch line 50 from the control unit 16 allows the instruction fetcher 8 to execute branches.

The instruction register 10 provides Source 1 and Source 2 register addresses to the register file 12 as Reg1 and Reg2. A result register address is provided as Dest. Opcode is provided to the control unit 16 along line 51. In addition, some instructions will provide a constant operand instead of encoding one or both source registers. The constant is provided by the constant unit 18. The instruction's source values are provided on Source 1 and Source 2 busses 52, 54 by the appropriate settings of the S1 Reg and S2 Reg signals at inputs E1, E2. The correct execution unit is enabled by providing the appropriate values for Pack Ops, Mem Ops and ALU Ops signals from the control unit 16 in accordance with the Opcode on line 51. The enabled unit will normally provide a result Res on a result bus 56. This is normally stored in the selected result register Dest in the register file 12. There are some exceptions to this.

4

Some instructions provide a double length result. These store the first part of the result in the normal way. In a subsequent additional stage, the second part of the result is stored in the next register in the register file 12 by asserting a Double signal 58.

Branches 50 need to read and adjust the instruction pointer 14. These cause the S1 Reg signal not to be asserted, and so the instruction pointer 14 provides the Source 1 value on line 60. The Source 2 value is provided in the normal way (either from a register in the register file 12, or the constant unit 18). The arithmetic unit 2 executes the branch calculations and its result is stored into the fetcher 8 on the New IP input 64, rather than the register file 12, signalled by the Branch line 50 from the control unit 16. This starts the fetcher from a new address.

Conditional branches must execute in two stages depending on the state of condition line 62. The first stage uses the Dest register as another source, by asserting a Read Dest signal 45. If the condition is satisfied, then the normal branch source operands are read and a branch is executed.

Calls must save a return address. This is done by storing the instruction pointer value in a destination register prior to calculating the branch target.

The computer described herein has several important qualities.

Source operands are always the natural word length. There can be one, two or three source operands.

The result is always the natural word length, or twice the natural word length. There is a performance penalty when it is twice the natural word length as it takes an extra stage to store and occupies two, rather than one, registers. For this computer, assume a natural word length of 64 bits. That is, each register in the register file has a predetermined capacity of 64 bits.

The execution units 2, 4, 6 do not hold any state between instruction execution. Thus subsequent instructions are independent.

## Non-Packed Instructions

The arithmetic unit 2 and memory access unit 4, along with the control unit 16 can execute the following instructions of a conventional instruction set. In the following definitions, a register is used to denote the contents of a register as well as a register itself as a storage location, in a manner familiar to a person skilled in the art.

mov	Move a constant or a register into a register.
add	Add two registers together and store the result in a third register (which could be the same as either of the sources)
sub	Subtract two registers and store the result in a third register
load	Use one register as an address and read from that location in memory, storing the result into another register
store	Use one register as an address and store the contents of another register into memory at the location specified by the address
cmpc	Compare two registers (or a register and a constant) for equality. If they are equal, store 1 into the destination register otherwise store zero
cmpge	Compare two registers (or a register and a constant) for orderability. If the second is not less than the first, store 1 into the destination register otherwise store zero
jump	Unconditional jump to a new location
jumpz	Jump to a new program location, if the contents of a specified register is zero

5,859,789

5

-continued

jumpz	Jump to a new program location, if the contents of a specified register is not zero
shr	Perform a bit-wise right shift of a register by a constant or another register and store the result in a destination register. The shift is signed because the sign bit is duplicated when shifting.
shl	Perform a bit-wise left shift of a register by a constant or another register and store the result in a destination register
or/xor	Perform a bit-wise logical operation (or/xor) on two registers and store result in destination register.

#### Packed Unit

FIG. 2 shows in a block diagram the packed arithmetic unit 6. This is shown as a collection of separate units each responsible for some subset of packed arithmetic instructions. It is quite probable that another implementation could combine the functions in different ways. The units include a byte replicate unit 70, a twist and zip unit 74, an obvious packed arithmetic unit 80, a multiply-add unit 76 and other packed arithmetic units 72,78. Only the multiply-add unit and obvious packed arithmetic unit are described in detail herein. These are operated responsive to a route opcode unit 82 which selectively controls the arithmetic units 70 to 80. Operands for the arithmetic units 70 to 80 are supplied along the Source 1 and Source 2 buses 52,54. Results from the arithmetic units are supplied to the result bus 56. The op input to the route opcode unit 82 receives the Pack Ops instruction from the control unit 16 (FIG. 1). It will be appreciated that the operands supplied on the Source 1 and Source 2 buses are loaded into respective input buffers of the arithmetic units and the results supplied from one or two output buffers to one or two destination registers in the register file 12.

#### Obvious Packed Arithmetic

The obvious packed arithmetic unit 80 performs operations taking the two source operands as containing several packed objects each and operating on respective pairs of objects in the two operands to produce a result also containing the same number of packed objects as each source. The operations supported can be addition, subtraction, comparison, multiplication, left shift, right shift etc. As explained above, by addressing a register using a single address an operand will be accessed. The operand comprises a plurality of objects which cannot be individually addressed.

FIG. 3 shows the symbols used in the diagrams illustrating the arithmetic units of the packed arithmetic unit 6.

FIG. 4 shows an obvious packed arithmetic unit which can perform addition, subtraction, comparison and multiplication of packed 16 bit numbers. As, in this case, the source and result bus widths are 64 bit, there are four packed objects, each 16 bits long, on each bus.

The obvious packed arithmetic unit 80 comprises four arithmetic logical units ALU0-ALU3, each of which are controlled by opcode on line 100 which is derived from the route opcode unit 82 in FIG. 3. The 64 bit word supplied from source register 1 SRC1 contains four packed objects S1[0]-S1[3]. The 64 bit word supplied from source register 2 SRC2 contains four packed objects S2[0]-S2[3]. These are stored in first and second input buffers 90,92. The first arithmetic logic unit ALU0 operates on the first packed object in each operand, S1[0] and S2[0] to generate a result R[0]. The second to fourth arithmetic logic units ALU1-ALU3 similarly take the second to fourth pairs of objects and provide respective results R[1] to R[3]. These

6

are stored in a result buffer 102. The result word thus contains four packed objects. An enable unit 101 determines if any of the unit should be active and controls whether the output buffer asserts its output.

The instructions are named as follows:

add2p	Add each respective S1[i] to S2[i] as 2's complement numbers producing R[i]. Overflow is ignored.
sub2p	Subtract each respective S2[i] from S1[i] as 2's complement numbers producing R[i]. Overflow is ignored.
cmp2p	Compare each respective S1[i] with S2[i]. If they are equal, set R[i] to all ones; if they are different, set R[i] to zero.
cmpge2ps	Compare each respective S1[i] with S2[i] as signed 2's complement numbers. If S1[i] is greater than or equal to S2[i] set R[i] to all ones; if S1[i] is less than S2[i] set R[i] to zero.
mul2ps	Multiply each respective S1[i] by S2[i] as signed 2's complement numbers setting R[i] to the least significant 16 bits of the full (32 bit) product.

Some obvious packed arithmetic instructions naturally take one packed source operand and one unpacked source operand. FIG. 5 shows such a unit.

The contents of the packed arithmetic unit of FIG. 5 are substantially the same as that of FIG. 4. The only different is that the input buffer 92' for the second source operand receives the source operand in unpacked form. The input buffer 92' receives the first source operand in packed form as before. One example of instructions using an unpacked source operand and a packed source operand are shift instructions, where the amount to shift by is not packed, so that the same shift can be applied to all the packed objects. Whilst it is not necessary for the shift amount to be unpacked, this is more useful.

shl2p	Shift each respective S1[i] left by S2 (which is not packed), setting R[i] to the result.
shr2ps	Shift each respective S1[i] right by S2 (which is not packed), setting R[i] to the result. The shift is signed, because the sign bit is duplicated when shifting.

It is assumed that the same set of operations are provided for packed 8 bit and packed 32 bit objects. The instructions have similar names, but replacing the "2" with a "1" or a "4".

#### Multiply-Add Unit

FIG. 6 shows the multiply-add unit 76. The multiply-add unit comprises two input buffers 104,106 which receive respective operands marked SRC1 and SRC2. In the illustrated embodiment, each operand comprises four packed 16 bit objects S1[0] to S1[3], S2[0] to S2[3]. A first multiplication circuit 108 receives the first object S1[0] from the first input buffer and the first object S2[0] in the second input buffer and multiplies them together to generate a first multiplication result. A second multiplication circuit 110 receives the second object S1[1] from the first buffer and the second object S2[1] from the second buffer and multiplies them together to generate a second multiplication result. A third multiplication circuit 112 receive the third objects S1[2],S2[2] from the first and second buffers and multiplies them together to generate a third multiplication result. A fourth multiplication circuit 114 receives the fourth objects S1[3],S2[3] from each buffer and multiplies them together to generate a fourth multiplication result. It will readily be appreciated that the multiplication circuits can take any

5,859,789

7

suitable form, well known to a person skilled in the art. The first multiplication result and second multiplication result are supplied to respective inputs of a first adder circuit 116. The third and fourth multiplication results are supplied to respective inputs of a second adder circuit 118. Each of the first and second adder circuits 116, 118 add together their respective inputs and supply the results to the input of a third adder circuit 120. The output of that adder circuit is held in an output buffer 122.

It will be appreciated that the multiplication operation carried out by the individual multiplication circuits will generate results having a "double length". That is, the multiplication together of two 16 bit objects will result in a 32 bit word. The addition of two 32 bit words will result in a word which has one or two bits more than 32 bits. This means that the capacity of the result buffer can safely be less than the capacity of one of the input buffers.

A type unit 124 receives opcode on line 118 derived from the route opcode unit 82 in FIG. 3. The type unit controls the output buffer 122.

The multiply-add unit is thus capable of executing a single instruction of the following form:

muladd2ps	multiply and add the packed 16-bit signed 2's complement objects.
-----------	---

The result of execution of that instruction will be to multiply together respective pairs of objects from two operands and to add together the results to provide a final result remaining within the original width of each operand. This allows the multiplication step to be carried out without incurring rounding errors, which normally happen as a result of multiplication steps to keep the word length within limits determined by the capacity of the available registers. The present multiply-add unit thus allows the multiplication to be performed at a high precision. Moreover, there is no need to incur rounding errors in the addition, because the length of the final result will inevitably be less than the capacity of one of the input buffers. As described earlier, the capacity of the input buffer will match the capacity of the available registers in the register file. Therefore, on execution of this instruction it can receive two operands, each occupying a single register and can guarantee that the result will occupy no more than one register. Conversely, because the capacity of the available register for the result is likely to be 64 bits, then it is certainly large enough to take the complete result and therefore prevent overflow areas from occurring.

It will readily be appreciated that it is possible to design similar multiply-add units for carrying out the combined multiply-add operation on different sizes of packed objects. It will also be readily appreciated that it is possible to hold the objects to be multiplied as part of a single operand in only one input buffer.

One example of use of the multiply-add unit is to evaluate the sum of products.

Sum of products is the evaluation of the following:

$$\sum_{i=0}^{N-1} A_i B_i$$

In the illustrated example of FIG. 6,  $N=4$ , operand SRC1 is  $A_1, A_2, A_3, A_4$  and operand SRC2 is  $B_1, B_2, B_3, B_4$ .

The multiply-add instruction can be used to effect this, and the sequence of instructions is shown in Annex A(i).

Another useful application of the multiply-add unit is to effect multiplication of a vector by a matrix. The vector is

8

represented by a vector operand comprising a plurality of objects each defined in vector elements. The matrix is represented by a set of matrix operands, each comprising a plurality of objects defining matrix elements. Each matrix operand is taken in turn and loaded into one of the input buffers of the unit of FIG. 6. The vector operand is loaded into the other input buffer. The multiply-add unit therefore multiplies respective pairs of vector elements and matrix elements to generate respective products and adds together the products. The result is held in the result buffer 122.

An exemplary instruction sequence for multiplying a vector by a matrix is shown in Annex A(ii).

#### Annex A(i)

```

15 ;sum of products of two vectors of N 16 bit objects
;R1 points to the first vector (A)
;R2 points to the second vector (B)
;R3 is the number of objects/4 in each vector
mov     R0, 0 ;clear accumulator
loop:
20   load     R4, R2 ;get 4 values from first vector
      load     R5, R2 ;get 4 values from second vector
      add     R2, R2, 8 ;increment vector pointer
      add     R3, R3, 8 ;increment vector pointer
      muladd2ps R6, R4, R5 ;multiply & add the 4 values
      add     R0, R0, R6 ;accumulate into running total
25   sub     R3, R3, 1 ;decrement counter
      jumpnz  R3, loop ;continue if incomplete
;the sum is in R0

```

#### Annex A(ii)

```

30 ;Vector times a matrix.
;Rvec contains the vector, 4 16bit objects
;Rmat0 to Rmat3 contain the transposed matrix,
; 4 x 4 16bit objects
;Rres produces the result, 4 16bit objects
muladd2ps temp0, Rvec, Rmat0 ;first result
35 muladd2ps temp1, Rvec, Rmat1 ;second result
muladd2ps temp2, Rvec, Rmat2 ;third result
muladd2ps temp3, Rvec, Rmat3 ;fourth result
;the separate results must now be packed together
;to create a vector of 4 16bit values. This is
;most naturally done with shifts, but as they are not
40 ;in this disclosure, I have used ands, shifts and ors,
;which requires more instructions.
and     temp0, temp1, 65535 ;mask result 1 overflow
and     temp1, temp1, 65535 ;mask result 2 overflow
and     temp2, temp2, 65535 ;mask result 3 overflow
;result 3 does not need masking
45 ;result 1 does not need shifting
shl     temp1, temp1, 16 ;shift result 1 up
shl     temp2, temp2, 32 ;shift result 2 up
shl     temp3, temp3, 48 ;shift result 3 up
or      temp0, temp0, temp1 ;combine result 1 & 2
or      temp2, temp2, temp3 ;combine result 2 & 3
or      Rres, temp0, temp2 ;combine
50 ;Rres now holds the result

```

What is claimed is:

1. An arithmetic unit for executing an instruction to multiply together pairs of objects from two sets of objects and to add together the resulting products, at least one set of said objects being represented by sub-strings forming a data string being a packed operand, the arithmetic unit comprising:

input buffer constructed and arranged to hold said set of objects forming said packed operand;

a plurality of multiplication circuits constructed and arranged to multiply together the respective pairs of objects, each said multiplication circuit including a pair of inputs for receiving the respective objects and including an output;

addition circuitry connected to receive the outputs of the multiplication circuits and constructed and arranged to

5,859,789

9

add together the resulting products of multiplication of the respective pairs of objects to generate a result; and an output buffer constructed and arranged to hold the result.

2. An arithmetic unit according to claim 1 wherein the addition circuitry comprises a first set of adder circuits each arranged to add together the outputs of a distinct two of said multiplication circuits, and a further adder circuit arranged to add together outputs of each of the first set of adder circuits to provide the result.

3. An arithmetic unit according to claim 1 or 2 wherein the input buffer comprises first and second input buffers each arranged to hold a respective one of the two set of objects.

4. An arithmetic unit according to claim 1 wherein the output buffer has a capacity which is less than the input buffer.

5. An arithmetic unit according to claim 1 wherein the input buffer is constructed and arranged to hold two sets of objects represented by sub-strings forming two data strings in form of two packed operands.

6. An arithmetic unit according to claim 1 wherein the input buffer is constructed and arranged to hold two sets of objects represented by sub-strings; wherein one set of sub-strings forms vector elements of a vector operand and the other set of sub-strings forms matrix elements of a matrix operand.

7. A computer comprising a processor, memory and data storage circuitry for holding data strings, wherein said processor comprises an arithmetic unit including

an input buffer constructed and arranged to hold two sets of objects, wherein at least one set of objects is represented by sub-strings of a data string forming said packed operand;

a plurality of multiplication circuits constructed and arranged to multiply simultaneously together respective pairs of objects from said two sets of objects, each said multiplication circuit including a pair of inputs for receiving said respective objects and including an output;

addition circuitry connected to receive the outputs of the multiplication circuits and constructed and arranged to add together the resulting products of multiplication of the respective of objects to generate a result; and

an output buffer constructed and arranged to hold the result, and wherein there is stored in said memory a sequence of instructions comprising at least an instruction to multiply together said pairs of objects from said two sets of objects and to add together the resulting products, said instruction being executed by the arithmetic unit.

8. A computer according to claim 7 wherein the data storage circuitry comprises a plurality of register stores each having a predetermined bit capacity matching the length of each of said data strings and being arranged to store said data strings as packed operands including objects represented by sub-strings.

10

9. A method of operating a computer to multiply together a vector and a matrix wherein the vector is represented by a vector data string comprising a plurality of sub-strings defining vector elements, the vector elements being objects arranged to form a packed operand, and wherein the matrix is represented by a set of matrix data strings each comprising a plurality of sub-strings defining matrix elements, the matrix elements being objects arranged to form packed operands, the method comprising selecting each of said matrix data strings in turn and executing for each selected data string a single instruction which:

loads the selected data string in form of packed source objects into an input buffer of an arithmetic unit;

loads said vector data string in form of packed source objects into said input buffer;

simultaneously multiplies respective pairs of vector elements and matrix elements of said data strings to generate respective products;

add together said products; and

generates a result.

10. A method of executing an instruction for multiplying together pairs of objects from two sets of objects and adding together the resulting products, said method including:

providing two sets of objects to an input buffer, wherein at least one said set of objects being represented by sub-strings forming a data string in form of a packed operand;

supplying said objects to a plurality of multiplication circuits for multiplying together respective pairs of said objects from said two sets, each said multiplication circuit receiving the respective objects defined by said sub-strings at a pair of inputs and providing an output; receiving the outputs of the multiplication circuits by addition circuitry for summing together the multiplication outputs to generate a result; and

holding in an output buffer the result.

11. A method according to claim 10 wherein said summing in the addition circuitry includes summing pairs of outputs from the multiplication circuits in a first set of adder circuits and then summing pairs of outputs of each of the first set of adder circuits to provide the result.

12. A method according to claim 10 wherein both said sets of objects are represented by sub-strings forming two data strings in form of two packed operands.

13. A method according to claim 12 wherein one of the two packed operands is a vector operand including a plurality of said objects defining vector elements and the other of the two packed operands is a matrix operand including a plurality of objects defining matrix elements.

\* \* \* \* \*

A-xi

42390P5943C



# *VIS™ Instruction Set User's Manual*

July 1997



Sun Microelectronics  
2550 Garcia Avenue  
Mountain View, CA 94043 U.S.A.  
1-800-681-8845  
[www.sun.com/sparc](http://www.sun.com/sparc)

Part Number: 805-1394-01

Copyright © 1997 Sun Microsystems, Inc. All Rights Reserved.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY EXPRESS REPRESENTATIONS OR WARRANTIES. IN ADDITION, SUN MICROSYSTEMS, INC. DISCLAIMS ALL IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

This document contains proprietary information of Sun Microsystems, Inc. or under license from third parties. No part of this document may be reproduced in any form or by any means or transferred to any third party without the prior written consent of Sun Microsystems, Inc.

Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The information contained in this document is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

Printed in the United States of America.

## 2.3 The UltraSPARC Front End

The UltraSPARC front end is essentially the Prefetch/Dispatch Unit (PDU). Figure 2-2 illustrates the major components of the UltraSPARC-I front end.

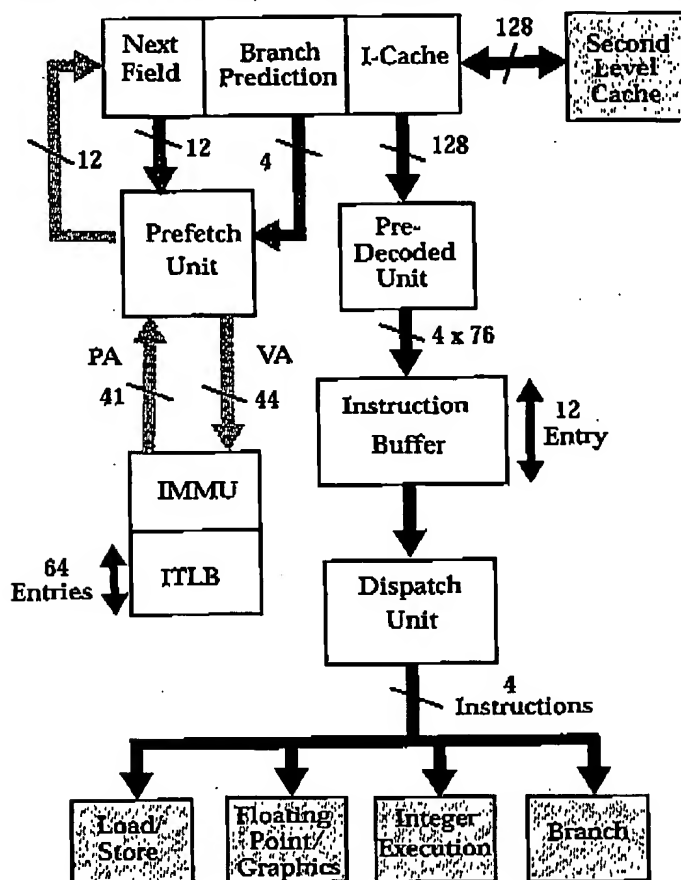


Figure 2-2 UltraSPARC-I Front End

Instructions are prefetched from a pseudo 2-way 16kbyte instruction cache. Each line in the I-Cache contains 8 instructions (32 bytes). Every pair of instructions has a 2-bit branch prediction field which maintains history of a possible branch in the pair. The four prediction states are the conventional strongly taken, likely taken, strongly not-taken and likely not-taken. The advantage of the in-cache prediction scheme is that it avoids the alias problems encountered in branch history

## 2. UltraSPARC Concepts

buffer and other similar structures. Every single branch in the I-Cache has its dedicated prediction bits (ignoring the rare case of branch couples), which translates into a successful prediction rate of 88% for integer code, 94% for floating-point (SPEC92) and 90% for typical database applications.

Every group of four instructions in the cache has a "next field" which is simply a pointer to where the prefetcher should access instructions for the very next cycle. In the case of sequential code or for code with a branch predicted not-taken, the next field points to the next 4 instructions in the cache. The next field will contain the I-Cache index (including the set) of the branch target if a branch is predicted taken. The advantage of this scheme is that the next field can always be fed back to the I-Cache without qualifying a possible branch. In order to provide a one-cycle loop back to the I-Cache, a fast dual-ported structure was used to implement the next field and the branch prediction bits. Only one set of the cache is accessed during a fetch, saving power and reducing the cache cycle time. Both tags are read so that an incorrect set prediction can be corrected. A two-cycle penalty occurs for a set misprediction. The next field mechanism allows UltraSPARC to speculate 5 branches deep representing up to 18 instructions.

Instructions prefetched by the PDU are expanded to 76 bits in order to facilitate decoding done by the grouping logic. These decoded instructions are forwarded to a 12-deep instruction buffer which allows the prefetcher to get ahead of the execution units. As long as the instruction queue is kept almost full, cache miss, set miss and micro-TLB (uTLB) miss penalties can be hidden from the execution units.

A single entry uTLB provides the prefetcher with a local copy of the last virtual-to-physical address translation. In the rare case of a uTLB miss a 1-cycle fetch penalty is incurred in order to get the address from the 64-entry fully associative instruction-TLB (iTLB).

The grouping logic always looks at the next four candidates in the instruction buffer and based on resource availability and dependencies, issues up to four instructions. Maintaining more than one Program Counter (PC) per group allows UltraSPARC to dispatch, in the same group, instructions from two adjacent basic blocks.

### 2.3.1 Integer Execution Unit (IEU)

The Integer Execution Unit (IEU) performs integer computation for all integer arithmetic/logical operations. The IEU as depicted in Figure 2-3 includes

*Sun Microsystems, Inc.*

9

## 2. UltraSPARC Concepts

A separate 64-bit adder is provided for virtual address additions for memory instructions. A simple 64-bit integer multiplier and divider complement the IEU. The multiplication unit implements a 2-bit Booth encoding algorithm with an "early-out" mechanism, with a typical latency of 8 clock cycles. A 1-bit non-restoring subtraction algorithm is used in the divide unit, which yields a latency of 67 clock cycles for a 64-bit by 64-bit division.

### 2.3.2 Floating Point/Graphics Unit (FGU)

The Floating-Point and Graphics Unit (FGU) as illustrated in Figure 2-4 integrates five functional units and a 32 registers by 64 bits Register File. The floating-point adder, multiplier and divider perform all FP operations while the graphics adder and multiplier perform the graphics operations of the VIS Instruction Set.

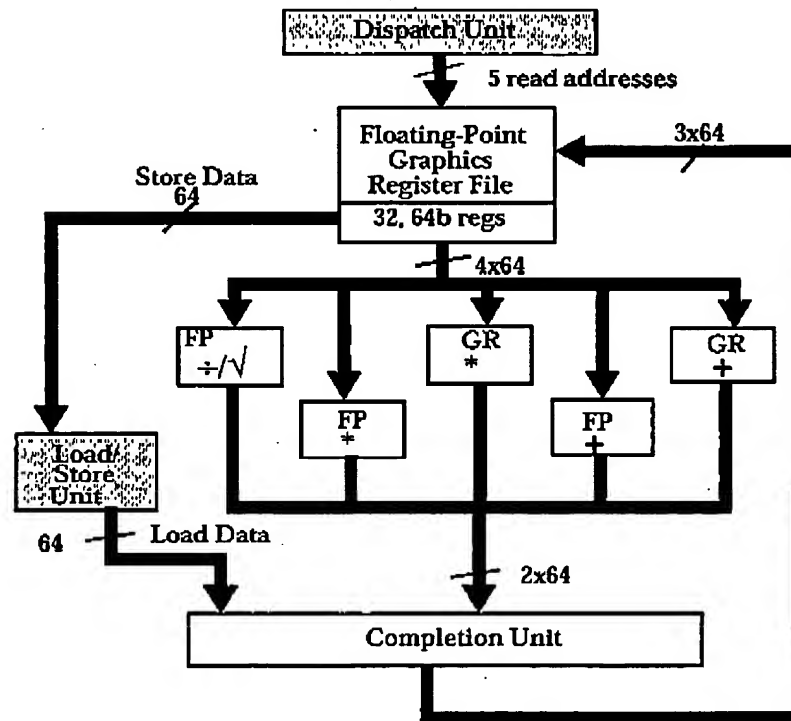


Figure 2-4 Floating Point and Graphics Unit

Sun Microsystems, Inc.

11

### *VIS Instruction Set User's Manual*

A maximum of two floating-point/graphics Operations (FGops) and one FP load/store operation are executed in every cycle (plus another integer or branch instruction). All operations, except for divide and square-root, are fully pipelined. Divide and square-root operations complete out-of-order without inhibiting the concurrent execution of other FGops. The two graphics units are both fully pipelined and perform operations on 8 or 16-bit pixel components with 16 or 32-bit intermediate results.

The Graphics Adder performs single cycle partitioned add and subtract, data alignment, merge, expand and logical operations. Four 16-bit adders are utilized and a custom shifter is implemented for byte concatenation and variable byte-length shifting. The Graphics Multiplier performs three cycle partitioned multiplication, compare, pack and pixel distance operations. Four 8x16 multipliers are utilized and a custom shifter is implemented. Eight 8-bit pixel subtractions, absolute values, additions and a final alignment are required for each pixel distance operation.

### **2.3.3 Load/Store Unit (LSU)**

The Load/Store Unit (LSU) executes all instructions that transfer data between the memory hierarchy and the Integer and Floating Point/Graphics Register files. The LSU includes the Data Cache, Load Buffer, Store Buffer, and is very closely coupled to the second level external cache. See Figure 2-5 for a functional diagram of the Load/Store Unit.

#### **2.3.3.1 Data Cache**

The Data Cache (D-Cache) is a 16kB, direct-mapped cache. It has a 32B (256 bits) line size, with 16B (128 bits) sub-blocks. It is virtually-indexed and physically-tagged. The D-Cache is non-blocking and operates using a write-through, no-write-allocate policy. Strict inclusion with respect to the E-cache is maintained, facilitating cache coherency. The D-Cache data SRAM is single-ported and can support a 64-bit load or a 64-bit store every cycle. In the event of a D-Cache miss, an entire sub-block (16B) can be written in one clock. The D-Cache tag SRAM has two ports, a read port and a write port. These two ports allow a load or store to perform a tag look-up in parallel with the allocation for an older D-Cache miss.

#### **2.3.3.2 Load Buffer**

The load buffer can eliminate stalls caused by D-Cache misses, load-after-store hazards, and other conflicts. Nine entries were implemented to cover the additional 6-cycle latency of a D-Cache miss/E-Cache hit. A rate of one load E-Cache

*Sun Microelectronics*

12

## 4. Using VIS

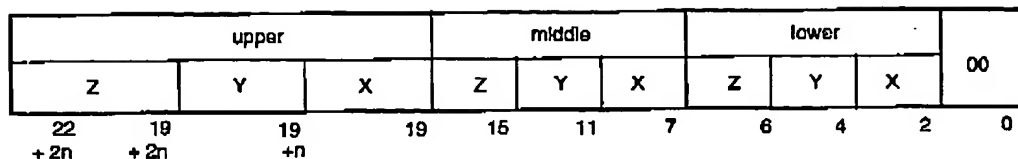


Figure 4-30 Three Dimensional Array Blocked-Address Format (Array32)

See the example on page 101, to see how the `array8`, the load and the add/sub instructions are used and grouped together for maximum throughput. The grouping takes into consideration the latencies of the different instructions i.e the load, `ldda`, following the `array8`, does not load the voxel just addressed by the `array8` in its grouping, but rather the voxel addressed by `array8` in the previous grouping.

The array instructions operate on all 64 bits of an integer register. Solaris 2.5 allows all 64 bits of the registers `%g2-%g4` and `%o0-%o7` to be used; other registers cannot be relied on to retain their upper 32 bits. Since the current SPARCCompiler 4.x has limited support for 64-bit integer operations, the array instructions might not be accessed efficiently from C. For a coding example, see "Using `array8` With Assembly Code" on page 101.

4.7.11 `vis_pdist()`

## Function

Compute the absolute value of the difference between two pixel pairs.  
i.e. between eight pairs of `vis_u8` components

## Syntax

```
vis_d64 vis_pdist(vis_d64 pixels1, vis_d64 pixels2, vis_d64
                  accumulator);
```

## Description

`vis_pdist()` takes three double-precision arguments `pixels1`, `pixels2` and `accum`. `pixels1` and `pixels2` contain 8 pixels each in raw format. The pixels are subtracted from one another, pair wise, and the absolute values of the differences are accumulated into `accum`. Note that the destination register is a double-precision floating-point register, which contains an integral value.

To use `vis_pdist()` from C, it is necessary for the accumulating register `accumulator` to appear both as an argument and as the receiver of the return value.

Sun Microsystems, Inc.

87

*VIS Instruction Set User's Manual*

The `vis_pdist()` instruction is intended to accelerate motion compensation to support real-time video compression in such applications as H.320 video conferencing.

**Example**

```
vl_d64 accum, pixels1, pixels2;

accum = vis_fzero();
accum = vis_pdist(pixels1, pixels2, accum);
```

**4.7.12 Block Load and Store Instructions****Function**

Transfer 64 bytes of data between memory and registers.

**Syntax**

The Block Load and Store instructions do not have a C interface and must be coded in assembly language. For assembly language syntax refer to section 13.6.4 in the *UltraSPARC-I User's Manual*.

**Description**

The block load instruction loads 64 bytes of data, with a block transfer, from a 64-byte aligned memory area into eight double-precision floating-point registers.

The block store instruction stores data, with a block transfer, from eight double-precision floating-point registers to a 64 byte aligned memory area.

**Example**

Note that the loop must be unrolled to achieve maximum performance. All FP registers are double-precision. Eight versions of this loop are needed to handle all the cases of double word misalignment between the source and destination.

```
loop:
  faligndata    %d0, %d2, %d34
  faligndata    %d2, %d4, %d36
  faligndata    %d4, %d6, %d38
  faligndata    %d6, %d8, %d40
  faligndata    %d8, %d10, %d42
  faligndata    %d10, %d12, %d44
  faligndata    %d12, %d14, %d46
  addcc        10, -1, 10
  bg,pt        11
  fmovd        %d14, %d48
```

*Sun Microelectronics*

88



*VIS Instruction Set User's Manual*

```

        accum = vis_faligndata(lookup, accum);
        lookup = vis_ld_u8_i((vis_ras) table, byte2);
        accum = vis_faligndata(lookup, accum);
        lookup = vis_ld_u8_i((vis_ras) table, byte1);
        accum = vis_faligndata(lookup, accum);
        lookup = vis_ld_u8_i((vis_ras) table, byte0);
        accum = vis_faligndata(lookup, accum);

        ((vis_d64 *) dst)[i] = accum;
    }
    break;
}

/* Update pointers, remaining width. */
src += 8*doubles;
dst += 8*doubles;
width -= 8*doubles;

/* Finish up any remaining pixels. */
for (i = 0; i < width; ++i)
    dst[i] = table[src[i]];
}

```

**5.2.4 Alpha Blending Two Images**

This example illustrates an application where two images are blended together. For each pair of corresponding pixels in two images "s1" and "s2", a corresponding pixel is read from a third control image "alpha", to compute:

$$\begin{aligned} \text{dst} &= (\text{alpha}/256)*\text{s1} + (1 - \text{alpha}/256)*\text{s2} \\ &= (\text{s1} - \text{s2})*(\text{alpha}/256) + \text{s1} \end{aligned}$$

Note that alpha can only range between 0 and 255, so strictly speaking we should divide it by 255, not 256. However, the division by 256 occurs for free when we perform the vis\_fnul8x16 operation, and the destination will differ from the correct result by at most 1. Whether this trade-off is acceptable or not depends on the application.

The following illustrates the processing of one scan line:

```

#define VIS_OFFSET(addr) ((addr) & 7)
#define VIS_ALIGN(addr) ((addr) & -7)
void
alpha_blend (vis_u8 *d, vis_u8 *s1, vis_u8 *s2, vis_u8 *a,
             int width)
/*
 * Arguments
 * d = pointer to destination data
 * s1 = pointer to data for image "s1"
 * s2 = pointer to data for image "s2"
 * a = pointer to data for control image alpha
 * width = data width of s1, s2 and alpha */

```

Sun Microelectronics

116

## 5. Advanced Topics

```

(
/* Last byte of destination. */
vis_u8 *d_end;

/* Doubleword-aligned pointers. */
vis_d64 *d_aligned, *s1_aligned, *s2_aligned, *alpha_aligned;

/* Alignment of original pointers. */
int d_offset, s1_offset, s2_offset, alpha_offset;

/* Unaligned data from memory. */
vis_d64 u_alpha_0, u_alpha_1, u_s1_0, u_s1_1, u_s2_0, u_s2_1;

/* Properly aligned data. */
vis_d64 quad_a, dbl_s1, dbl_s2, dbl_a, dbl_d;

/* Temporaries. */
vis_d64 dbl_s1_e, dbl_s2_e, dbl_tmpl, dbl_tmpl2;
vis_d64 dbl_sum1, dbl_sum2;

/* Edge mask for partial stores. */
unsigned int emask;

/* Loop variables. */
int i, times;

vis_write_gsr(3 << 3);
/* Four (= 7 - 3) bits of fractional precision. */

d_end = d + width - 1;
d_offset = VIS_OFFSET(d);
d_aligned = (vis_d64 *) VIS_ALIGN(d);

/* Compute initial edge mask for destination. */
emask = vis_edge8(d, d_end);

/* Align addresses relative to destination alignment and
load data. */
s1_offset = VIS_OFFSET(s1 - d_offset);
s1_aligned = vis_alignaddr(s1, - d_offset);
u_s1_0 = s1_aligned[0];
u_s1_1 = s1_aligned[1];

s2_offset = VIS_OFFSET(s2 - d_offset);
s2_aligned = vis_alignaddr(s2, - d_offset);
u_s2_0 = s2_aligned[0];
u_s2_1 = s2_aligned[1];

off_a = VIS_OFFSET(a - d_offset);
alpha_aligned = vis_alignaddr(a, - d_offset);
u_alpha_0 = alpha_aligned[0];
u_alpha_1 = alpha_aligned[1];

```

Sun Microsystems, Inc.

117

*VIS Instruction Set User's Manual*

```

/* Number of times through the loop. */
times = ((vis_u32) d_end >> 3) - ((vis_u32) d_aligned >> 3) + 1;

for (i = 0; i < times; ++i) {
    (void) vis_alignaddr((void *) 0, off_a);
    /* Set alignment for alpha. */
    quad_a = vis_faligndata(u_alpha_0, u_alpha_1);
    u_alpha_0 = u_alpha_1;
    u_alpha_1 = alpha_aligned[i + 2];

    (void) vis_alignaddr((void *) 0, s1_offset);
    /* Set alignment for s1. */
    dbl_s1 = vis_faligndata(u_s1_0, u_s1_1);
    u_s1_0 = u_s1_1;
    u_s1_1 = s1_aligned[i + 2];

    (void) vis_alignaddr((void *) 0, s2_offset);
    /* Set alignment for s2. */
    dbl_s2 = vis_faligndata(u_s2_0, u_s2_1);
    u_s2_0 = u_s2_1;
    u_s2_1 = s2_aligned[i + 2];

    dbl_s1_e = vis_fexpand(vis_read_hi(dbl_s1));
    dbl_s2_e = vis_fexpand(vis_read_hi(dbl_s2));
    dbl_tmp2 = vis_fpsub16(dbl_s2_e, dbl_s1_e);
    dbl_tmp1 = vis_fmuls16(vis_read_hi(quad_a), dbl_tmp2);
    dbl_sum1 = vis_fpadd16(dbl_s1_e, dbl_tmp1);

    dbl_s1_e = vis_fexpand(vis_read_lo(dbl_s1));
    dbl_s2_e = vis_fexpand(vis_read_lo(dbl_s2));
    dbl_tmp2 = vis_fpsub16(dbl_s2_e, dbl_s1_e);
    dbl_tmp1 = vis_fmuls16(vis_read_lo(quad_a), dbl_tmp2);
    dbl_sum2 = vis_fpadd16(dbl_s1_e, dbl_tmp1);

    dbl_d = vis_freg_pair(vis_fpack16(dbl_sum1),
                          vis_fpack16(dbl_sum2));

    vis_pst_8(dbl_d, (void *) d_aligned, emask);
    ++d_aligned;

    emask = vis_edges8(d_aligned, d_end);
}
}

```

A-xii

42390P5943C



US005721697A

**United States Patent** [19]

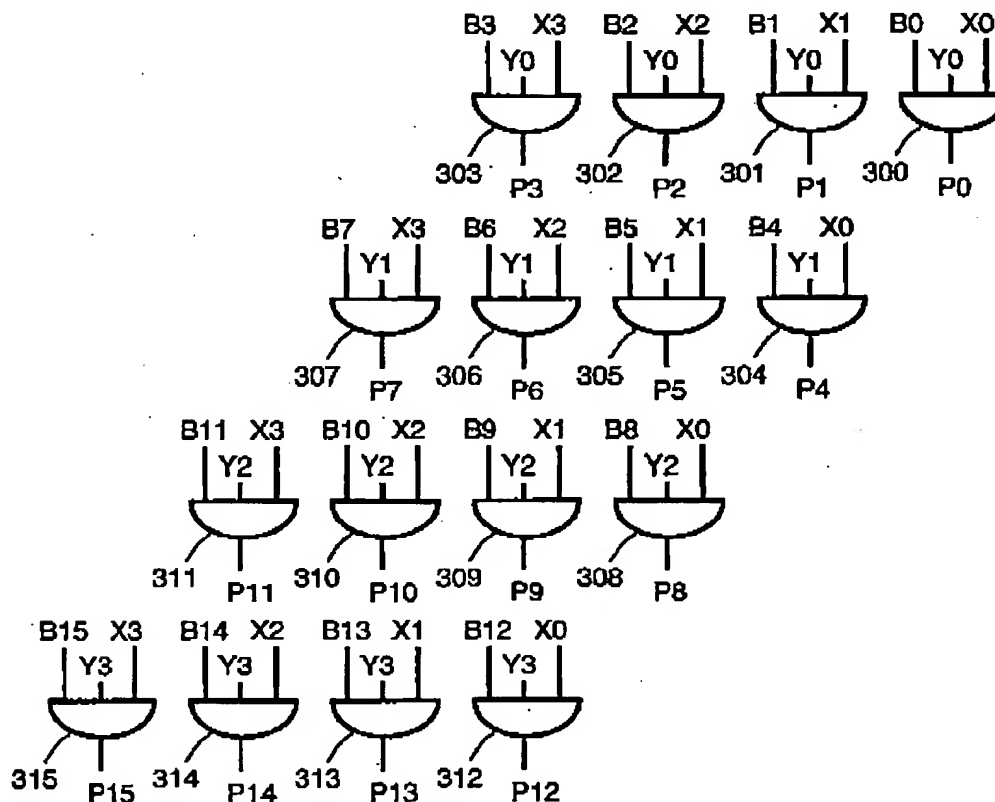
Lee

[11] Patent Number: **5,721,697**[45] Date of Patent: **Feb. 24, 1998****[54] PERFORMING TREE ADDITIONS VIA MULTIPLICATION**4,736,335 4/1988 Barica ..... 364/758  
5,095,457 3/1992 Jeong ..... 364/758[75] Inventor: **Ruby Bei-Loh Lee, Los Altos Hills, Calif.***Primary Examiner—Chuong Dinh Ngo*[73] Assignee: **Hewlett-Packard Company, Palo Alto, Calif.****[57] ABSTRACT**[21] Appl. No.: **649,349**[22] Filed: **May 17, 1996****Related U.S. Application Data**

[60] Provisional application No. 60/000,272, Jan. 16, 1995.

[51] Int. Cl.<sup>6</sup> ..... **G06F 7/50; G06F 7/52**[52] U.S. CL. .... **364/754; 364/758; 364/784; 364/786**[58] Field of Search ..... **364/754, 757, 364/758, 759, 760, 768, 784, 786****[56] References Cited****U.S. PATENT DOCUMENTS**

4,369,500 1/1983 Fette ..... 364/758

**16 Claims, 4 Drawing Sheets**

U.S. Patent

Feb. 24, 1998

Sheet 1 of 4

5,721,697

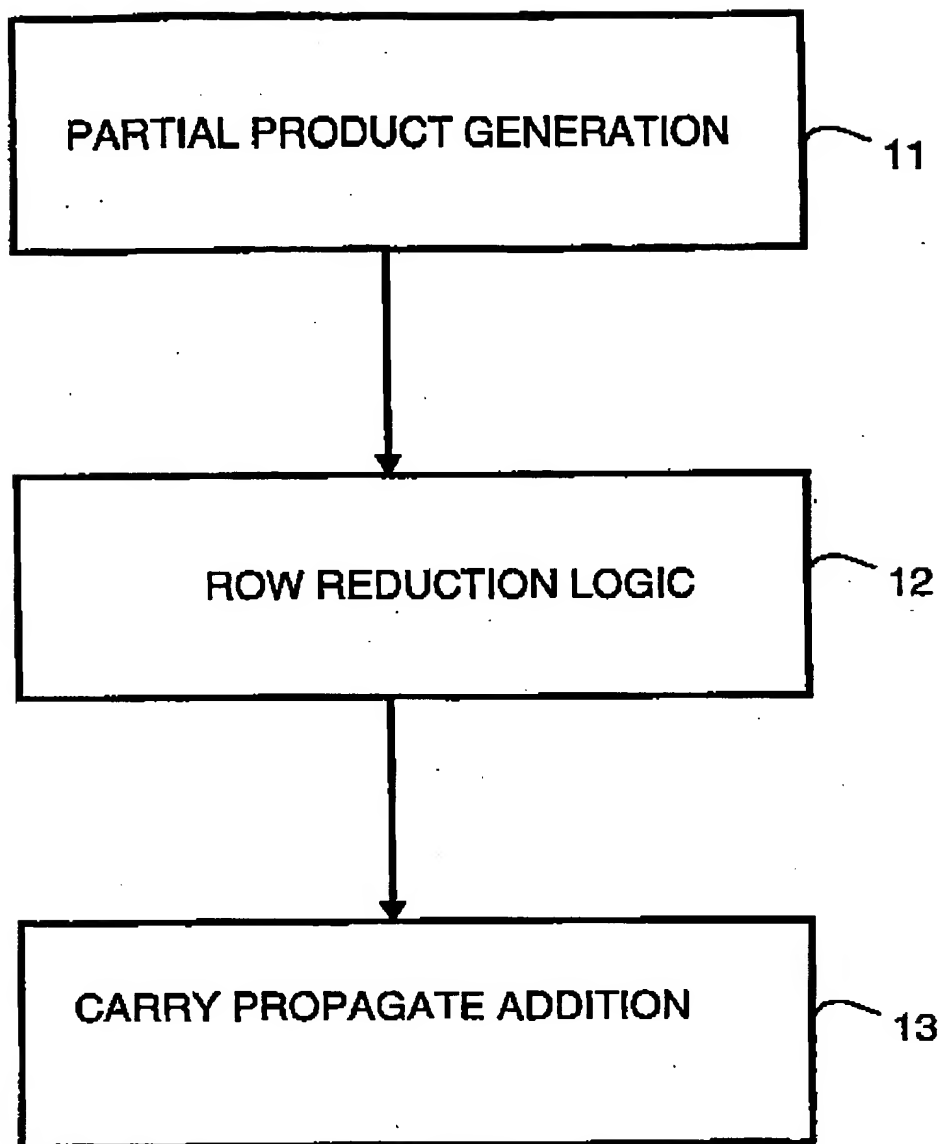


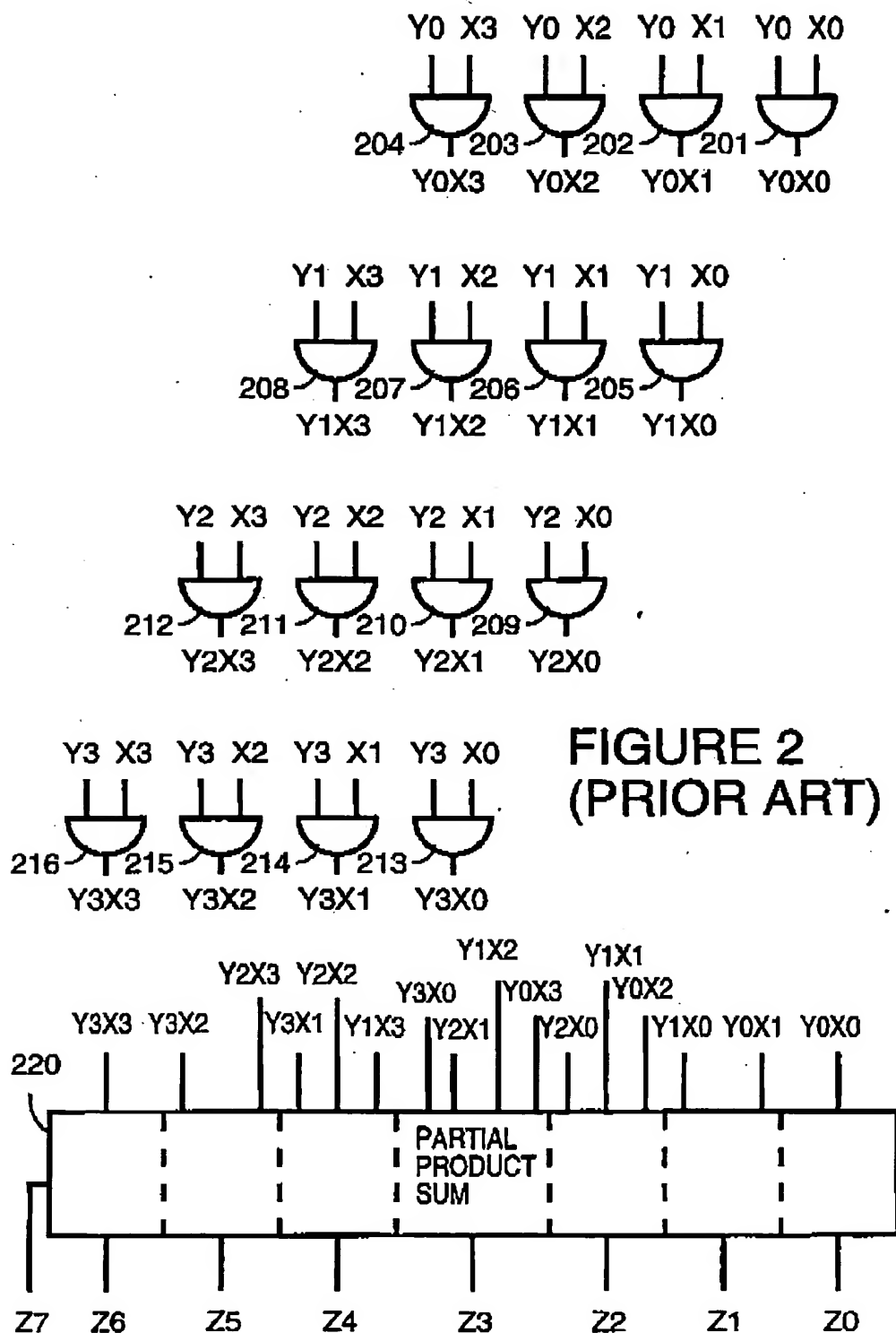
FIGURE 1

U.S. Patent

Feb. 24, 1998

Sheet 2 of 4

5,721,697



U.S. Patent

Feb. 24, 1998

Sheet 3 of 4

5,721,697

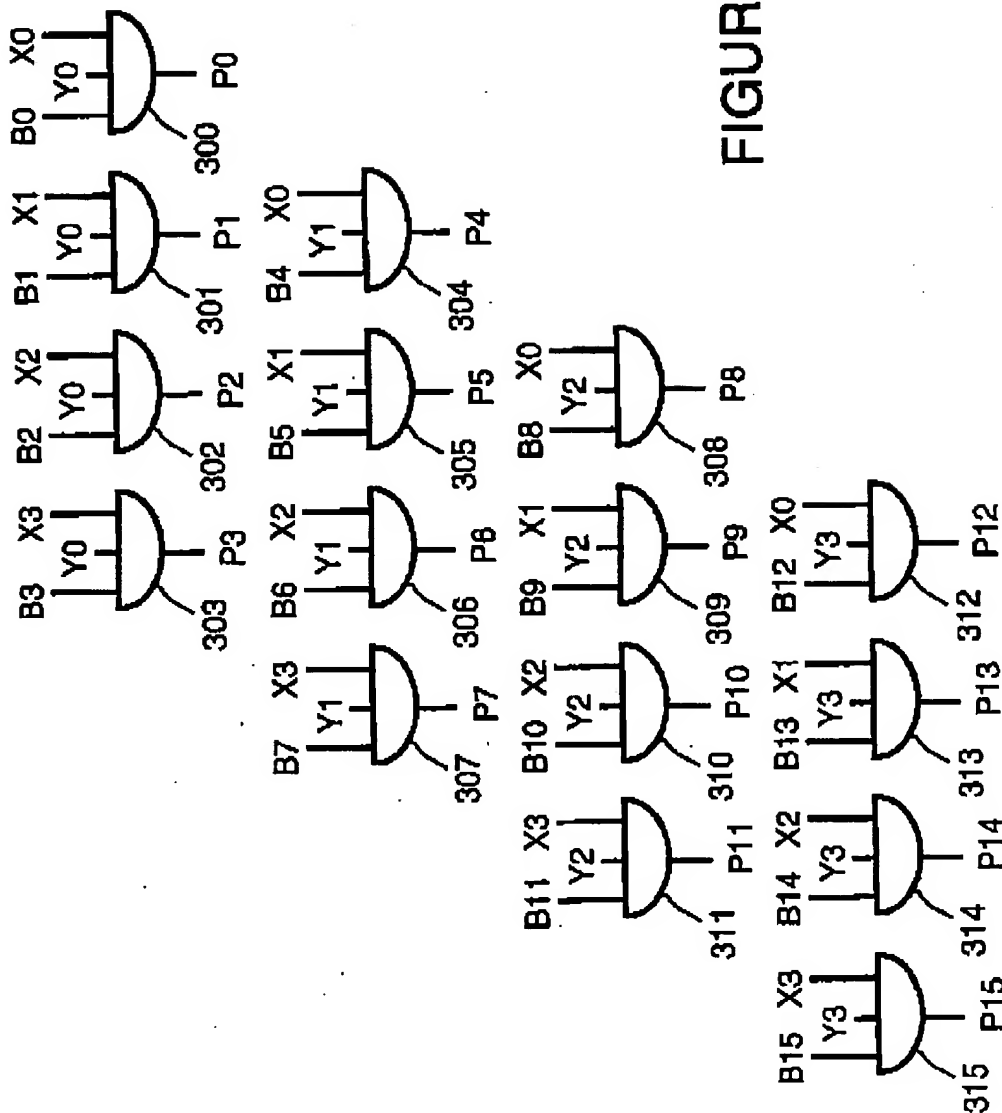


FIGURE 3



U.S. Patent

Feb. 24, 1998

Sheet 4 of 4

5,721,697

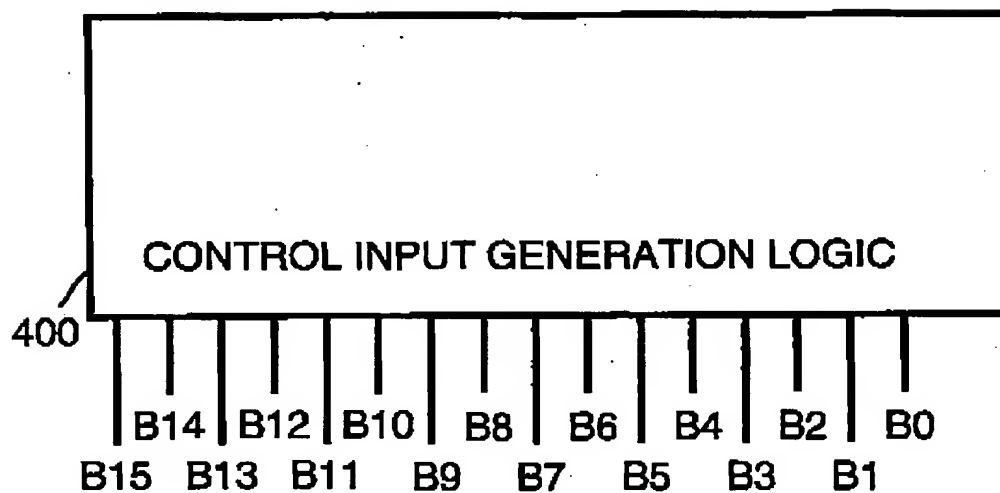


FIGURE 4

5,721,697

1

PERFORMING TREE ADDITIONS VIA  
MULTIPLICATIONCROSS REFERENCE TO RELATED  
APPLICATION

This application is based on provisional application Ser. No. 60/000,272, filed Jan. 16, 1995.

## BACKGROUND

The present invention concerns computer operations implemented in hardware and particularly hardware which performs tree additions.

In computer systems one or more arithmetic logic units (ALUs) are generally utilized to perform arithmetic operations. In addition to ALUs, high performance computing often include other special hardware to expedite the performance of specific tasks. For example, a computing system may include hardware devoted to performing multiplication and/or hardware devoted to performing division.

Complex operations for which there is no devoted hardware are generally implemented by a series of instructions. For example, a tree add operation is useful for video compression. In one case of a tree add instruction, four sixteen-bit half words originally in a single sixty-four bit register are added together. In another case of a tree add instruction, eight bytes originally in a single sixty-four bit register are added together. In another case of a tree add instruction, four bytes originally in a single thirty-two bit register are added together. And so on.

In order to perform a tree add instruction, it is generally required to place each operand in a separate register and then, to successively use the add operation implemented by the ALU to add operands together, two at a time. As will be understood, such an execution of a tree add instruction will generally take a large number of instruction cycles. As long as tree additions are rare, this is not a significant hindrance to high performance in a computing system. However, for a computing system which frequently performs tree additions, for example for video compression, implementing the tree add using a large number of instruction cycles could have a negative impact on overall system performance.

## SUMMARY OF THE INVENTION

In accordance with the preferred embodiment of the present invention, a multiplier is modified to perform a tree addition. A first value is input to the multiplier in place of a first multiplicand. The first value is a concatenation of addends upon which the tree addition is performed. A second value is input into the multiplier in place of a second multiplicand. Each bit of the second value is at logic zero except for a first subset of bits. The first subset of bits are bits of the second value, starting with the low order bit, which are at intervals equal to a bit length of each addend. Each of the first subset of bits is set to logic one. In partial product rows in the multiplier which correspond to the first subset of bits, certain partial products are forced to logic zero. This is done in such a way that all the addends for the tree addition are aligned in columns of the multiplier. The partial products are then summed to produce a result.

In the preferred embodiment, the partial products are generated using three-input logic AND gates. Particular partial products are forced to zero by plating a zero on a control input of the three-input logic AND gate used to generate the partial product.

Also, in the preferred embodiment, the partial products are summed in two steps. In a first step, the partial products are reduced into two rows of partial products. A carry propagate addition is then performed on the two rows of partial products to produce the result.

2

The present invention allows for a implementation of a tree addition with only minor changes to a multiplier.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a simplified block diagram of a multiplier in accordance with the prior art.

FIG. 2 shows a block diagram of a multiplier in accordance with the prior art.

FIG. 3 shows a simplified block diagram of circuitry which generates partial products for a modified multiplier in accordance with the preferred embodiment of the present invention.

FIG. 4 shows a simplified block diagram of circuitry which generates control inputs used to generate partial products for a modified multiplier in accordance with the preferred embodiment of the present invention.

## DESCRIPTION OF THE PRIOR ART

FIG. 1 shows a block diagram of an integer or mantissa multiplier. Partial product generation logic 11 generates rows of partial products. Row reduction logic 12 uses three-to-two counters to reduce the rows of partial products to two rows. A three-to-two counter is implemented using a one-bit adder slice which adds three one-bit inputs to produce a two-bit output. Carry propagate addition logic 13 performs a full carry-propagate add on the two remaining rows to produce a final product.

FIG. 2 shows a four-bit multiplier in accordance with the prior art. The multiplier multiplies a four-bit first multiplicand  $X_3X_2X_1X_0$  (base 2) with a four-bit second multiplicand  $Y_3Y_2Y_1Y_0$  (base 2) to produce an eight-bit result  $Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$  (base 2). As is understood by those skilled in the art, logic AND gates 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215 and 216 may be used to generate partial products for the multiplication. A partial product sum circuit 220 sums the partial products generated by logic AND gates 201 through 216 to produce the result. Partial product sum circuit includes both reduction logic and carry propagate addition logic, as described above.

The two multiplicands,  $X_3X_2X_1X_0$  and  $Y_3Y_2Y_1Y_0$ , the partial products generated by logic AND gates 201 through 216, and the result produced by partial product sum circuit 220 may be placed in a table in such a way as to summarize operation of the multiplier. For example, such a table is shown as Table 1 below:

TABLE 1

				$X_3$	$X_2$	$X_1$	$X_0$	
				$Y_0X_3$	$Y_0X_2$	$Y_0X_1$	$Y_0X_0$	$Y_0$
			$Y_1X_3$	$Y_1X_2$	$Y_1X_1$	$Y_1X_0$		$Y_1$
		$Y_2X_3$	$Y_2X_2$	$Y_2X_1$	$Y_2X_0$			$Y_2$
$Y_3X_3$	$Y_3X_2$	$Y_3X_1$	$Y_3X_0$					$Y_3$
$Z_7$	$Z_6$	$Z_5$	$Z_4$	$Z_3$	$Z_2$	$Z_1$	$Z_0$	

In the notation used in Table 1 above, the bit position of each bit of both multiplicands and the result is specifically identified. Additionally, the bits of the multiplicand which are used to form each partial product are specifically set out. As is understood by those skilled in the art, the information shown in Table 1 above may be set out using abbreviated or simplified notation, as in Table 2 below:

**5,721,697**

3

TABLE 2

				$X_0$	$X_2$	$X_1$	$X_3$	
			$X_3$	$X_1$	$X_2$	$X_0$	$X_3$	$Y_0$
		$X_0$	$X_2$	$X_1$	$X_0$			$Y_1$
	$X_2$	$X_1$	$X_0$	$X_3$				$Y_2$
$Z_7$	$Z_6$	$Z_5$	$Z_4$	$Z_3$	$Z_2$	$Z_1$	$Z_0$	$Y_3$

In Table 2 above, each row of partial products is shown without the Y component. Thus, the first row of partial products is listed in Table 2 as follows:

$\Sigma, \Sigma_2, X_1, X_0$

However, this is a simplified notation which represents the following partial products:

$$Y_0X_3, Y_0X_2, Y_0X_1, Y_0X_0$$

Similarly, the last row of partial products listed in Table 2 represents the following partial products:

$$Y_3X_3, Y_3X_2, Y_3X_1, Y_3X_0$$

Using the simplified notation of Table 2, an eight-bit multiplier may be described as shown in Table 3 below:

TABLE 3

									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_0$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_1$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_2$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_3$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_4$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_5$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_6$
									$X_7$	$X_8$	$X_5$	$X_4$	$X_3$	$X_2$	$X_1$	$X_0$	$Y_7$
$Z_{13}$	$Z_{14}$	$Z_{13}$	$Z_{12}$	$Z_{11}$	$Z_{10}$	$Z_9$	$Z_8$	$Z_7$	$Z_6$	$Z_5$	$Z_4$	$Z_3$	$Z_2$	$Z_1$	$Z_0$		

The multiplier shown in Table 3 multiplies an eight-bit first multiplicand  $X_7X_6X_5X_4X_3X_2X_1X_0$  (base 2) with an eight-bit second multiplicand  $Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0$  (base 2) to produce an sixteen-bit result  $Z_{15}Z_{14}Z_{13}Z_{12}Z_{11}Z_{10}Z_9Z_8Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$  (base 2). To further simplify notation, the partial products and the sixteen-bit result may be written without subscripts. Thus, the eight-bit multiplication may be represented as in Table 4 below:

TABLE 4

[illegible]

### DESCRIPTION OF THE PREFERRED EMBODIMENT

Generally, most of the circuitry and execution latency of a multiplier exists in row reduction logic 12 and carry propagate addition logic 13. In the preferred embodiment, no changes are made to row reduction logic 12 or carry propagate additional logic 13 of a multiplier in order to perform a tree addition. But in partial product generation logic 11, the two input logic AND gates are replaced with three input logic AND gates.

FIG. 3 shows that in partial product generation logic 11, the two input logic AND gates are replaced with three input logic AND gates. The multiplier multiplies a four-bit first multiplicand  $X_3X_2X_1X_0$  (base 2) with a four-bit second multiplicand  $Y_3Y_2Y_1Y_0$  (base 2) to produce an eight-bit result  $Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$  (base 2). As is understood by those skilled in the art, logic AND gates 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314 and 315 may be used to generate partial products for the multiplication. When generating partial products for multiplication, control inputs  $B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}$  and  $B_{15}$  are all set to logic 1 in order to generate partial products  $P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}$  and  $P_{15}$ .

In addition, selection of multiplicands and selection of values of control inputs  $B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}$  and  $B_{15}$  can be used to perform a tree add as is further described below.

Particularly, when a tree add is to be performed on a plurality of addends within a first register, a first value in the first register is input in place of the first multiplicand for the multiplier. The first value is a concatenation of the addends. A second value is input into the multiplier in place of the second multiplicand. Each bit of the second value is at logic zero except for a first subset of bits. The first subset of bits includes a low order bit of the second value, and includes bits of the second value which, starting from the low order





5,721,697

9

products in the partial product rows are forced to logic zero, so that addends for the tree addition are aligned in columns of the multiplier.

7. A multiplier as in claim 1 wherein when a tree add is to be performed on a plurality of addends:

a first value is input into the multiplier in place of a first multiplicand, the first value being a concatenation of the addends;

a second value is input into the multiplier in place of a second multiplicand, each bit of the second value being at logic one; and,

a portion of the partial products in the multiplier are forced to logic zero, so that addends for the tree addition are aligned in columns of the multiplier.

8. A method for using a multiplier to perform a tree addition comprising the steps of:

(a) inputting a first value to the multiplier in place of a first multiplicand, the first value being a concatenation of addends upon which the tree addition is performed;

(b) inputting a second value to the multiplier in place of a second multiplicand, each bit of the second value being at logic zero except for a first subset of bits comprising bits of the second value which, starting with the low order bit, are at intervals equal to a bit length of each addend, each of the first subset of bits being set to logic one;

(c) for partial product rows in the multiplier which correspond to the first subset of bits, forcing to logic zero a portion of partial products in the partial product rows, so that addends for the tree addition are aligned in columns of the multiplier; and,

(d) summing the partial products to produce a result.

9. A method as in claim 8 wherein in step (c) partial products are forced to zero by placing a zero on a control input of three-input logic AND gate used to generate the partial product.

10. A method as in claim 8 wherein step (d) includes the following substeps:

10

(d.1) reducing the partial products into two rows of partial products; and,

(d.2) performing a functional equivalent of a carry propagate addition on the two rows of partial products to produce the result.

11. A method for using a multiplier to perform a tree addition comprising the steps of:

(a) inputting a first value to the multiplier in place of a first multiplicand, the first value including addends upon which the tree addition is performed;

(b) forcing a subset of the partial products to zero when performing a tree addition; and,

(c) summing the partial products to produce a result.

12. A method as in claim 11 wherein in step (b) partial products are forced to zero by placing a zero on a control input of three-input logic AND gate used to generate the partial product.

13. A method as in claim 11 wherein step (c) includes the following substeps:

(c.1) reducing the partial products into two rows of partial products; and,

(c.2) performing a functional equivalent of a carry propagate addition on the two rows of partial products to produce the result.

14. A method as in claim 11 wherein in step (a) the first value is a concatenation of the addends.

15. A method as in claim 11 additionally including the following step:

inputting a second value to the multiplier in place of a second multiplicand, each bit of the second value being at logic one.

16. A method as in claim 15 wherein in step (b) partial products are forced to zero by placing a zero on a control input of three-input logic AND gate used to generate the partial product.

\* \* \* \* \*

A-xiii

**THIS PAGE BLANK (USPTO)**

42390P5943C



US005938756A

# United States Patent [19]

Van Hook et al.

[11] Patent Number: 5,938,756

[45] Date of Patent: \*Aug. 17, 1999

## [54] CENTRAL PROCESSING UNIT WITH INTEGRATED GRAPHICS FUNCTIONS

[75] Inventors: Timothy J. Van Hook, Menlo Park; Leslie Dean Kohn; Robert Yung, both of Fremont, all of Calif.

[73] Assignee: Sun Microsystems, Inc., Palo Alto, Calif.

[\*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: 08/635,350

[22] Filed: Apr. 19, 1996

### Related U.S. Application Data

[63] Continuation of application No. 08/236,572, Apr. 29, 1994, abandoned.

[51] Int. Cl.<sup>6</sup> ..... G06F 15/00

[52] U.S. Cl. .... 712/23; 345/530; 345/502

[58] Field of Search ..... 395/800, 376, 395/595, 375, 800.23; 712/23, 200, 245; 345/506, 502, 503, 215

### [56] References Cited

#### U.S. PATENT DOCUMENTS

4,965,752	10/1990	Keith	364/522
5,081,698	1/1992	Kohn	395/122
5,157,388	10/1992	Kohn	340/800
5,241,636	8/1993	Kohn	395/375
5,254,979	10/1993	Trevett et al.	345/113
5,268,995	12/1993	Diefendorff et al.	395/122
5,394,515	2/1995	Lentz et al.	395/115
5,533,185	7/1996	Lentz et al.	395/162
5,546,551	8/1996	Kohn	395/375
5,560,035	9/1996	Garg et al.	395/800.23
5,574,872	11/1996	Rotem et al.	395/376
5,592,679	1/1997	Yung	396/800.23
5,594,880	1/1997	Moyer et al.	395/595
5,604,909	2/1997	Joshi et al.	395/384

### FOREIGN PATENT DOCUMENTS

0 303 752 A1	2/1989	European Pat. Off.	
0 306 305 A2	3/1989	European Pat. Off.	
0 395 293 A1	10/1990	European Pat. Off.	
0 395 348 A3	10/1990	European Pat. Off.	G06F 9/30
0 380 098 A3	8/1991	European Pat. Off.	G06F 15/80
0 485 776 A3	5/1992	European Pat. Off.	
2 265 065	9/1993	United Kingdom	G06F 15/70
WO 88/07235	9/1988	WIPO	
94/03287	4/1994	WIPO	G06F 9/38

### OTHER PUBLICATIONS

Mahon et al. "Hewlett-Packard Precision Architecture: The Processor" pp. 4-22, Aug. 1986.

(List continued on next page.)

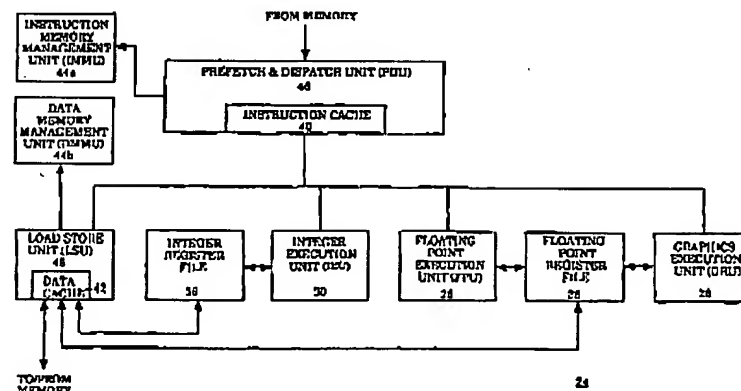
Primary Examiner—Larry D. Donaghue

Attorney, Agent, or Firm—Townsend and Townsend and Crew LLP

### [57] ABSTRACT

The integer execution unit (IEU) of a central processing unit (CPU) is provided with a graphics status register (GSR) for storing a graphics data scaling factor and a graphics data alignment address offset. Additionally, the CPU is provided with a graphics execution unit (GRU) for executing a number of graphics operations in accordance to the graphics data scaling factor and alignment address offset, the graphics data having a number of graphics data formats. In one embodiment, the GRU is also used to execute a number of graphics data addition, subtraction, rounding, expansion, merge, alignment, multiplication, logical, compare, and pixel distance operations. The graphics data operations are categorized into a first and a second category, and the GRU concurrently executes one graphics operations from each category. Furthermore, under this embodiment, the IEU is also used to execute a number of graphics data edge handling and 3-D array addressing operations, while the load and store unit (LSU) of the CPU is also used to execute a number of graphics data load and store operations, including conditional store operations.

9 Claims, 20 Drawing Sheets





5,938,756

Page 2

## OTHER PUBLICATIONS

"i860 Microprocessor Architecture"; Intel. Neal Margulis; Osborne McGraw-Hill 1991, pp. 8-10, pp. 171-175, pp. 182-183.

Robert Tobias, "The LR33020 GraphX Processor: A Single Chip X-Terminal Controller," ISI Logic Corporation, IEEE, Feb. 24, 1992, p. 358 to p. 363.

O. Awsienko et al., "Boundary Aligner," IBM Technical Disclosure Bulletin, Dec. 1984, vol. 27, No. 7B, pp. 4247-4248.

"MC88110—Second Generation RISC Microprocessor User's Manual," Motorola 1991, pp. 1-4 to 1-17; 3-1 to 3-2/15; 5-1 to 5-25; 7-9; 7-14, 7-15; 7-19; 7-20, 7-21.

U.S. Patent

Aug. 17, 1999

Sheet 1 of 20

5,938,756

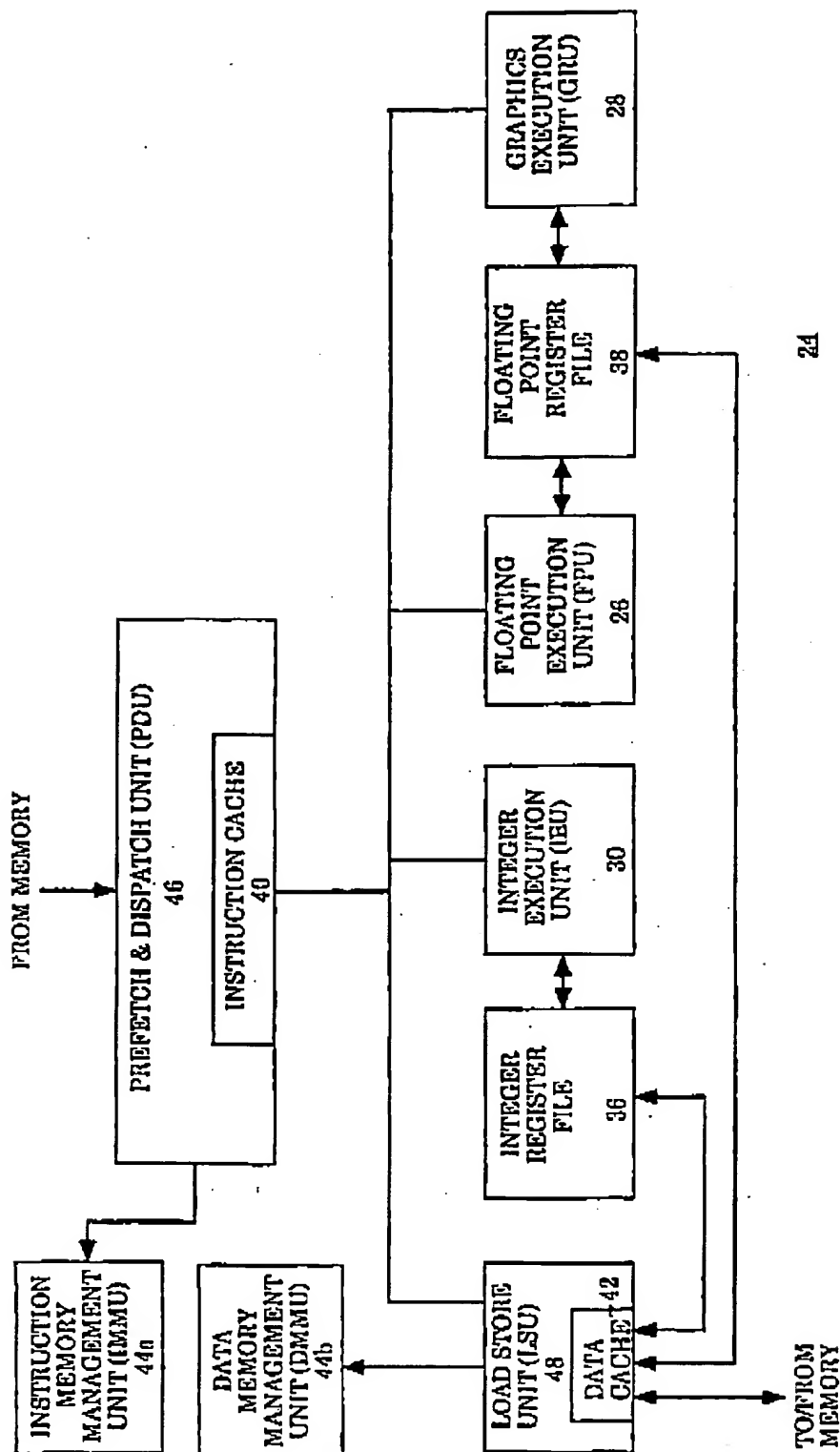


Figure 1

U.S. Patent

Aug. 17, 1999

Sheet 2 of 20

5,938,756

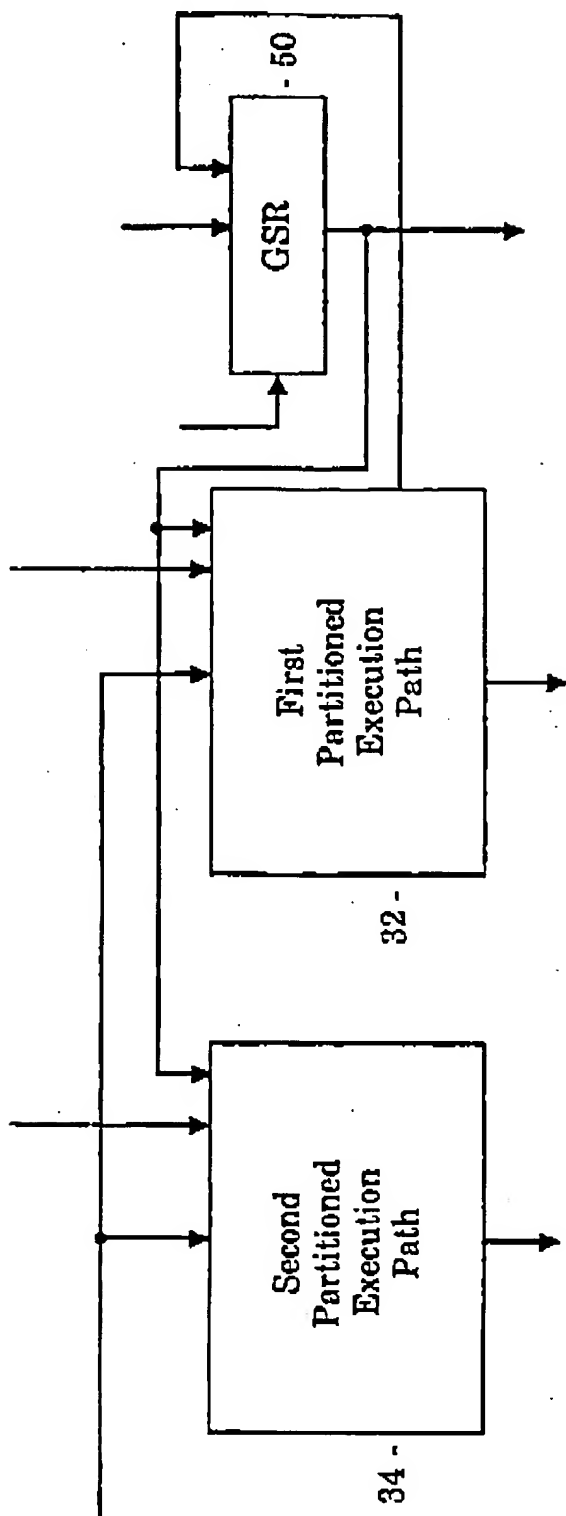


Figure 2

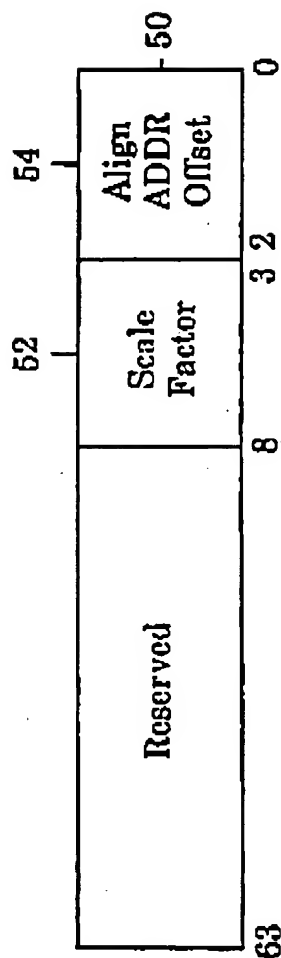


Figure 3

U.S. Patent

Aug. 17, 1999

Sheet 3 of 20

5,938,756

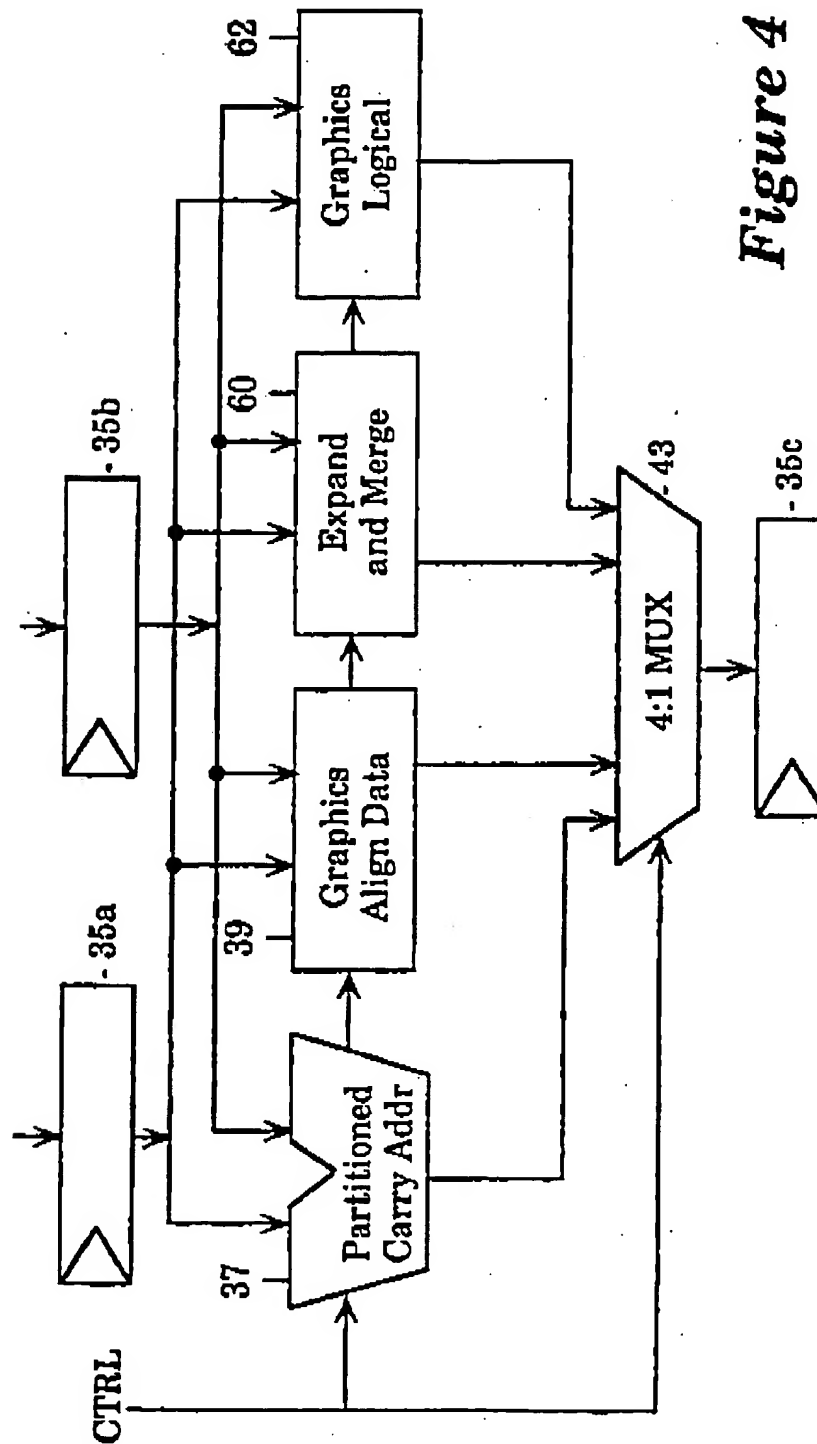


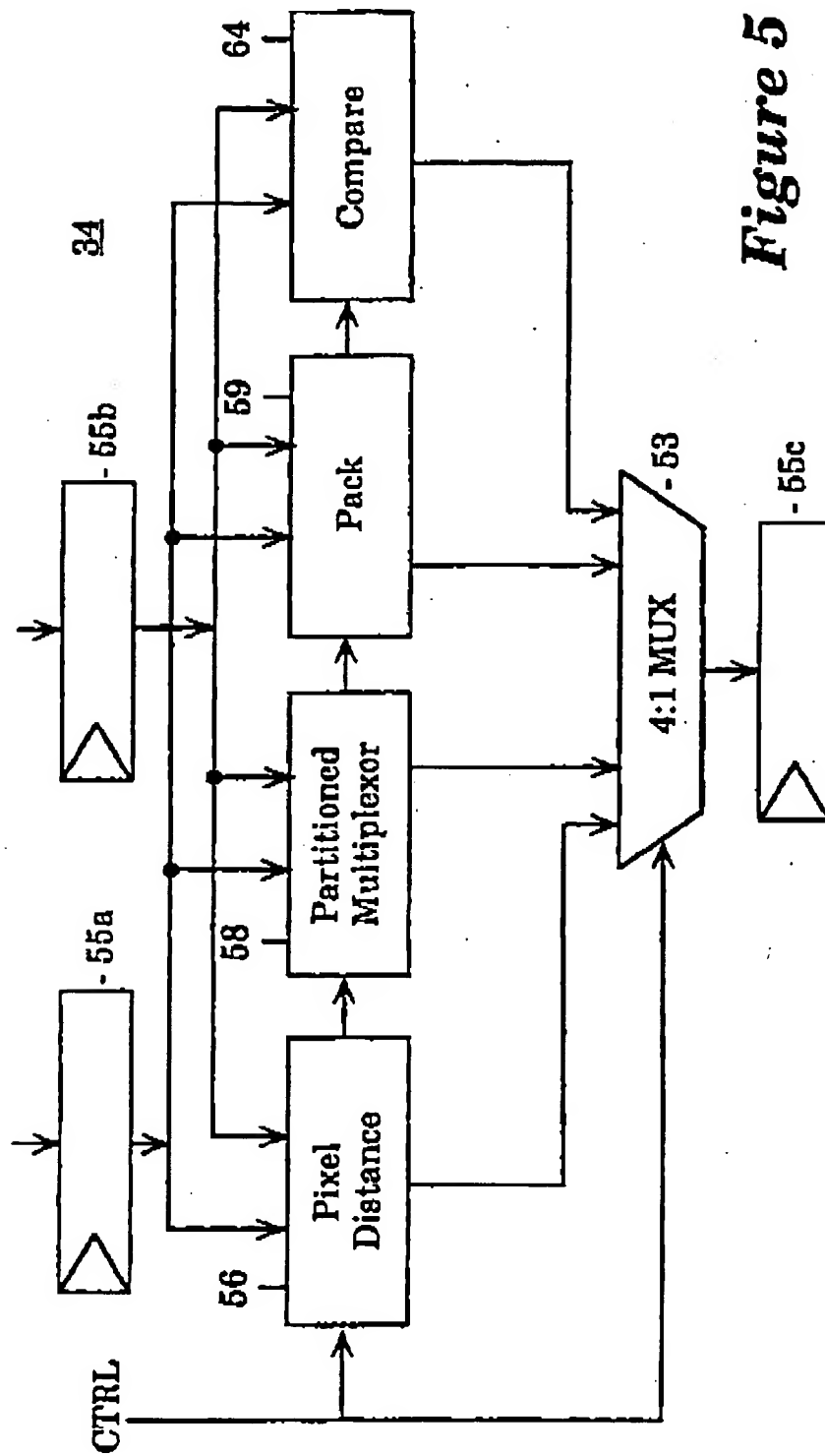
Figure 4

U.S. Patent

Aug. 17, 1999

Sheet 4 of 20

5,938,756

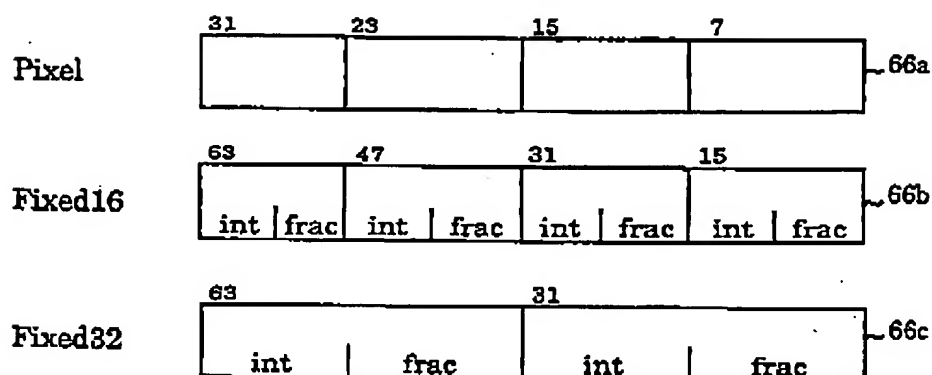
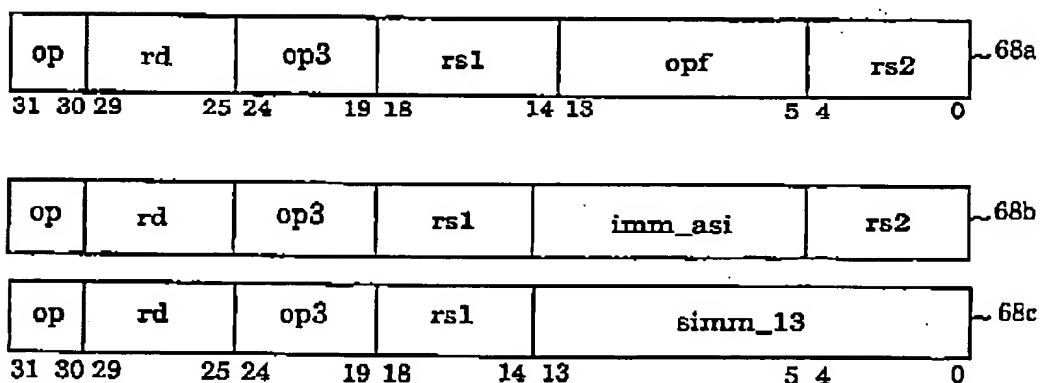
**Figure 5**

U.S. Patent

Aug. 17, 1999

Sheet 5 of 20

5,938,756

*Figure 6a**Figure 6b*

U.S. Patent

Aug. 17, 1999

Sheet 6 of 20

5,938,756

GRAPHIC INSTRUCTION GROUP	GRAPHIC INSTRUCTIONS
200 ~ GSR	RDASR, WRASR
202 ~ PARTITIONED ADD/SUBTRACT	FPADD, FPSUB
204 ~ GRAPHICS DATA ALIGNMENT	ALIGNADDRESS, ALIGNDATA
206 ~ PIXEL DISTANCE	PDIST
208 ~ PARTITIONED MULTIPLICATION	FPMULT
210 ~ PARTITIONED EXPAND & MERGE	FPEXPAND, FPMERGE
212 ~ PARTITIONED PACK	FPPACK
214 ~ LOGICAL	FZERO, FONE, FSRC, FNOT, FOR, FAND, FNAND, ETC.
216 ~ COMPARE	FPCMP
218 ~ EDGE HANDLING	EDGE
220 ~ 3-D ARRAY ACCESS	ARRAY
222 ~ MEMORY ACCESS	PST, SLD/SSST, BLD/TEST

Figure 6c

U.S. Patent

Aug. 17, 1999

Sheet 7 of 20

5,938,756

op = 10 , op3 = 110110

	opcode	op3	operation
98a~	ALIGNADDRESS	000011000	calculate address for misaligned data access
98b~	ALIGNADDRESS1_LITTLE	000011010	calculate address for misaligned data access, little endian
100~	FALIGNDATA	001001000	performs data alignment for misaligned data

Exemplary Assembly Language Syntax			
alignaddr	reg <sub>rs1</sub>	reg <sub>rs2</sub>	reg <sub>rd</sub>
alignaddrl	reg <sub>rs1</sub>	reg <sub>rs2</sub>	reg <sub>rd</sub>
faligndata	freg <sub>rs1</sub>	freg <sub>rs2</sub>	freg <sub>rd</sub>

Figure 7a

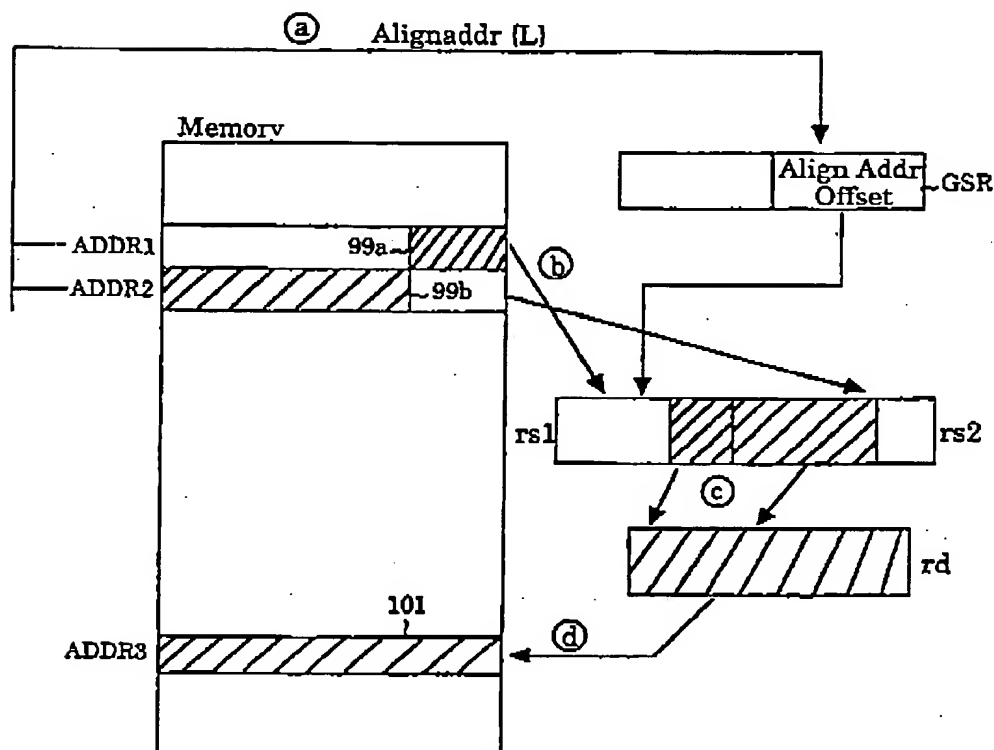


Figure 7b

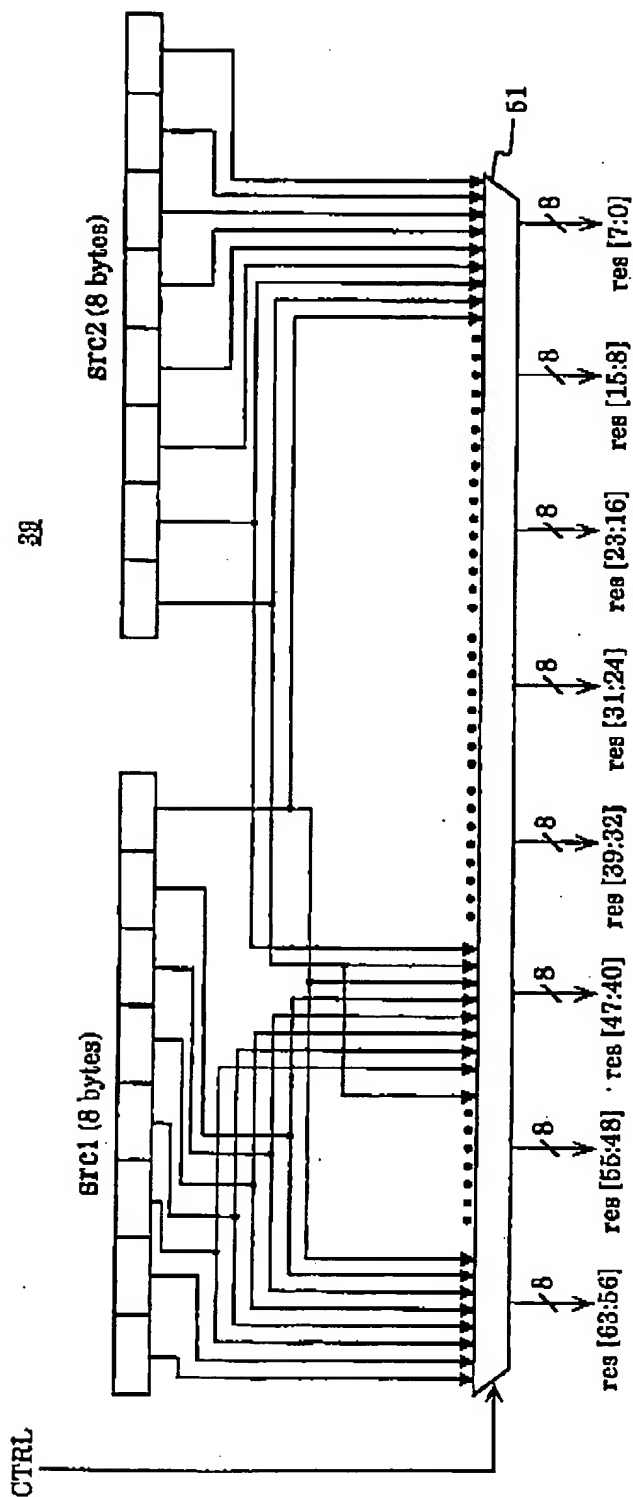


U.S. Patent

Aug. 17, 1999

Sheet 8 of 20

5,938,756

*Figure 7c*

U.S. Patent

Aug. 17, 1999

Sheet 9 of 20

5,938,756

op = 10 . op3 = 110110

	opcode	op3	operation
106a ~	FPACK16	000111001	4 16-bit add
106b ~	FPACK32	000111010	2 32-bit add
106c ~	FPACKFIX	000111101	4 16-bit subtract

Exemplary Assembly Language Syntax	
fpack16	reg <sub>rs2</sub> , reg <sub>rd</sub>
fpack32	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
fpackfix	reg <sub>rs2</sub> , reg <sub>rd</sub>

Figure 8a

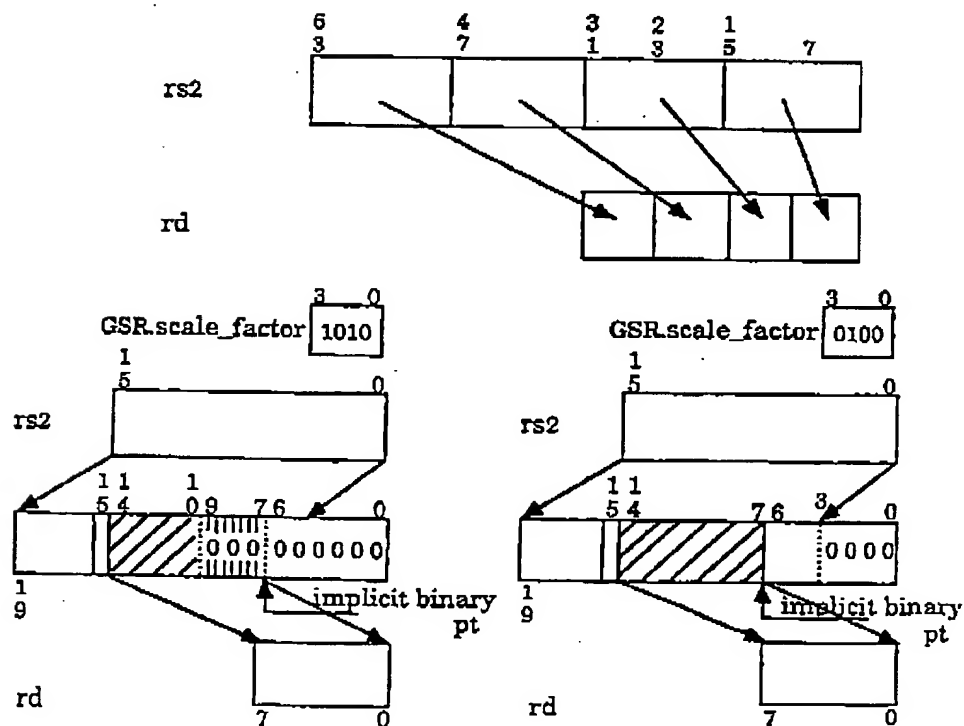


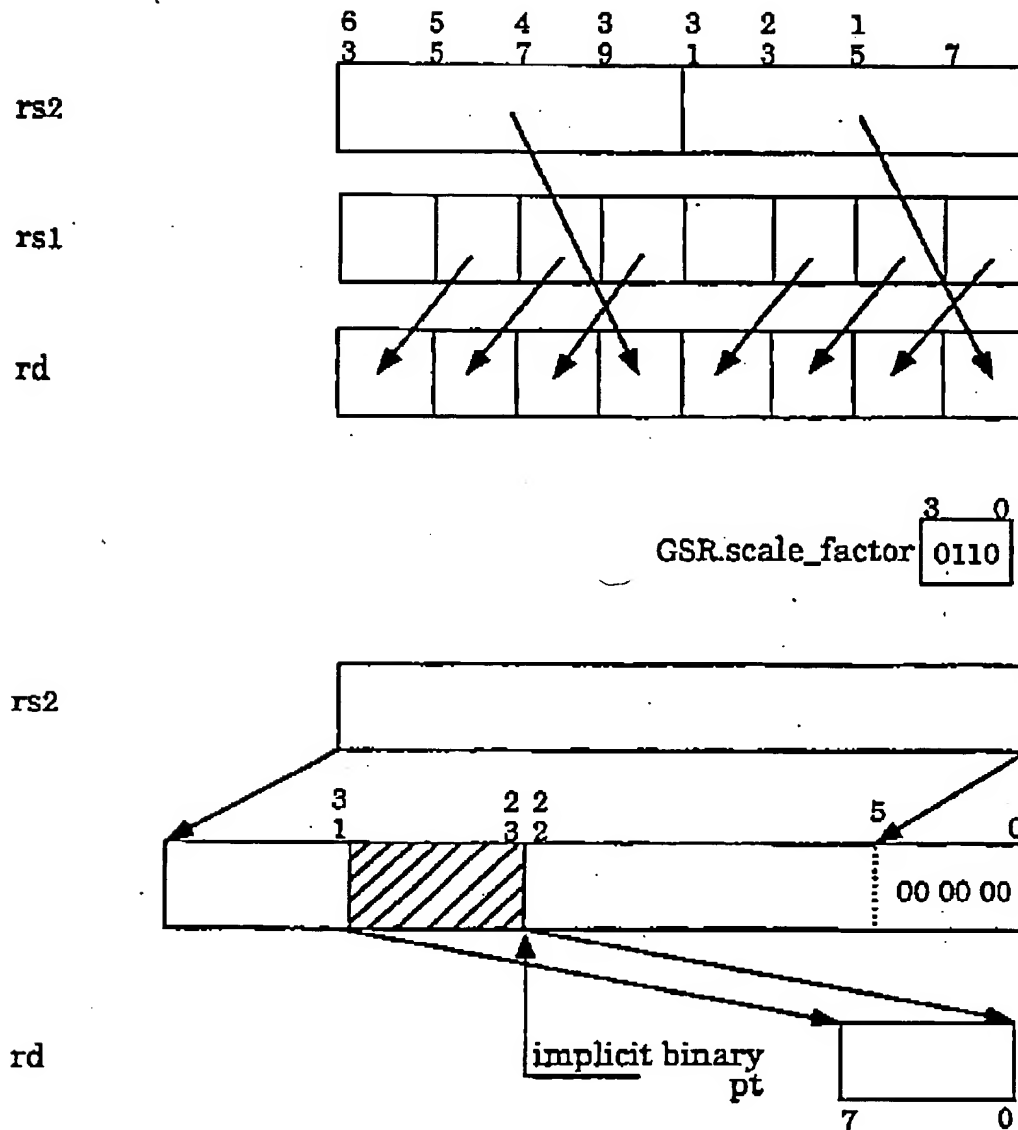
Figure 8b

U.S. Patent

Aug. 17, 1999

Sheet 10 of 20

5,938,756

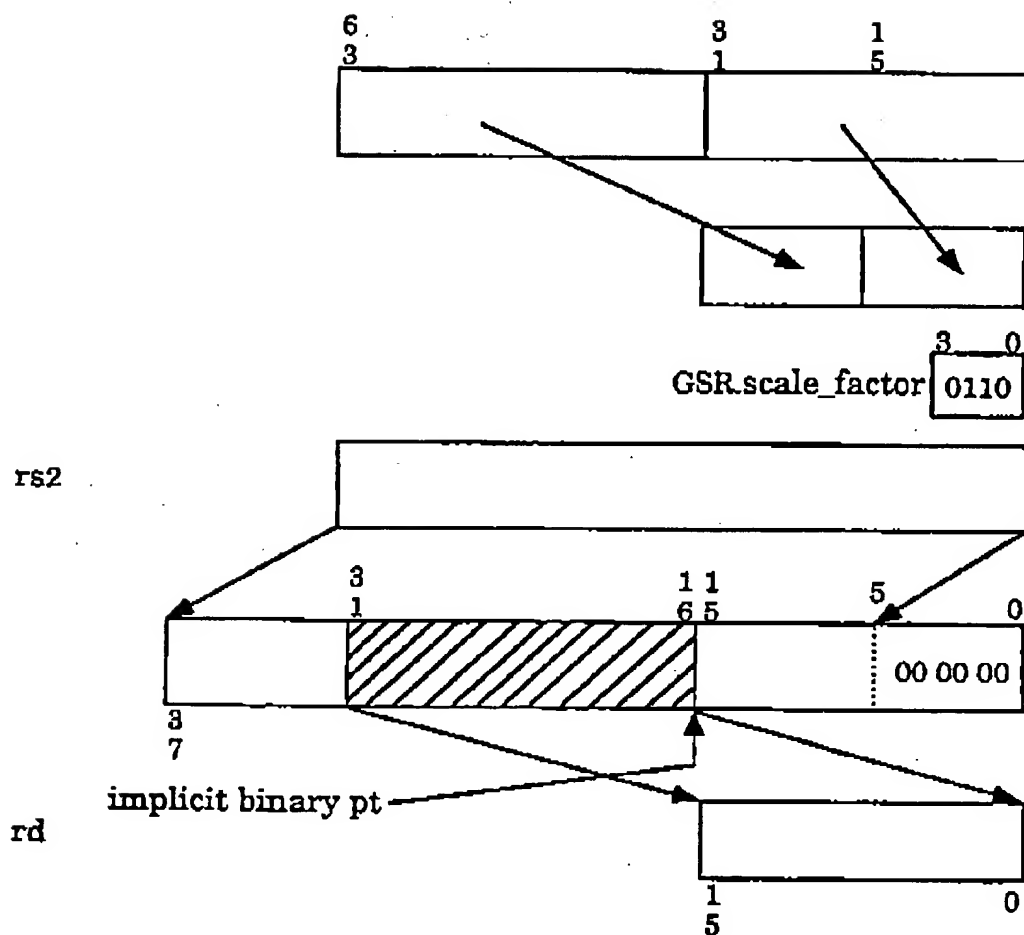
*Figure 8c*

U.S. Patent

Aug. 17, 1999

Sheet 11 of 20

5,938,756

*Figure 8d*

U.S. Patent

Aug. 17, 1999

Sheet 12 of 20

5,938,756

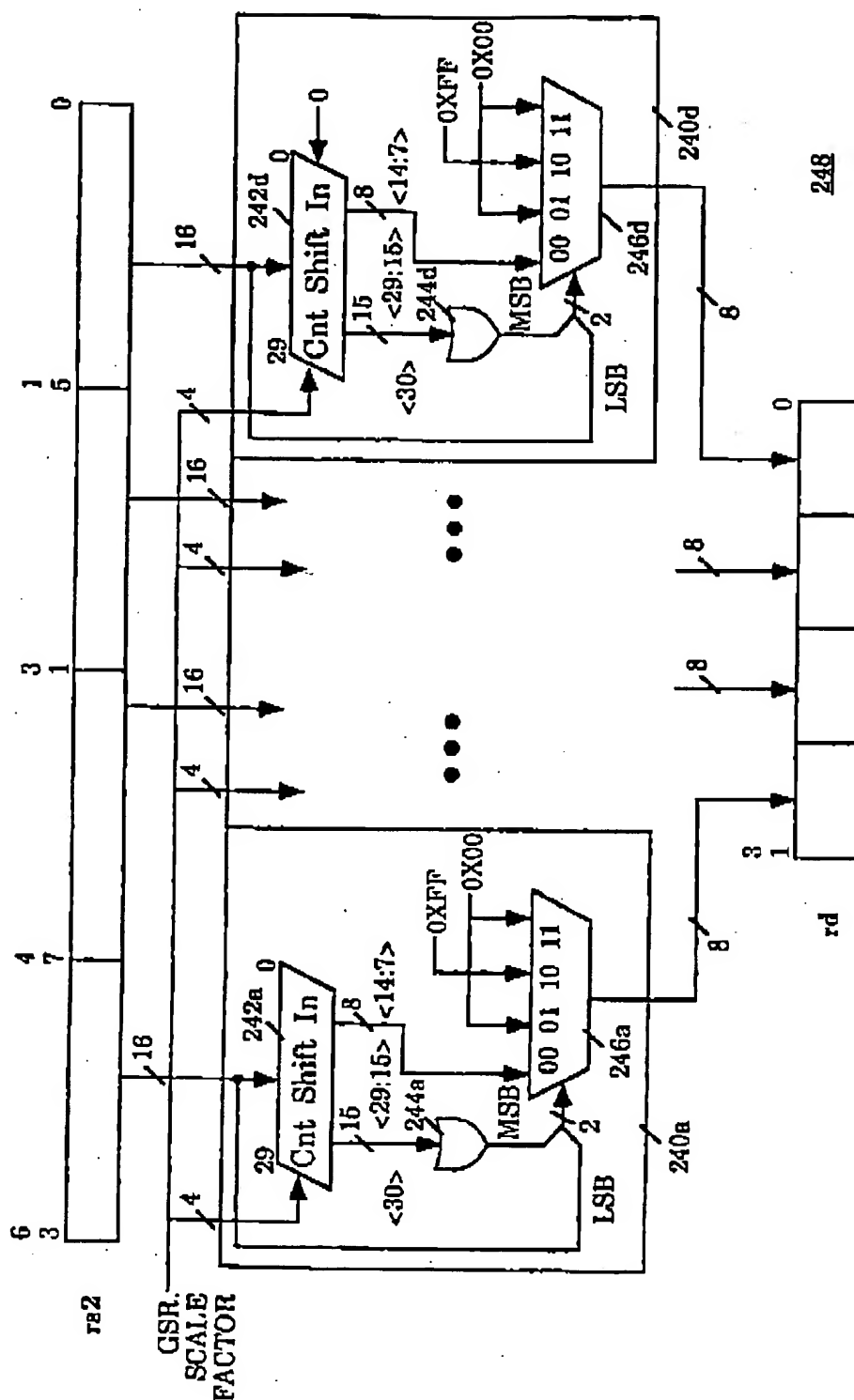


Figure 8e

U.S. Patent

Aug. 17, 1999

Sheet 13 of 20

5,938,756

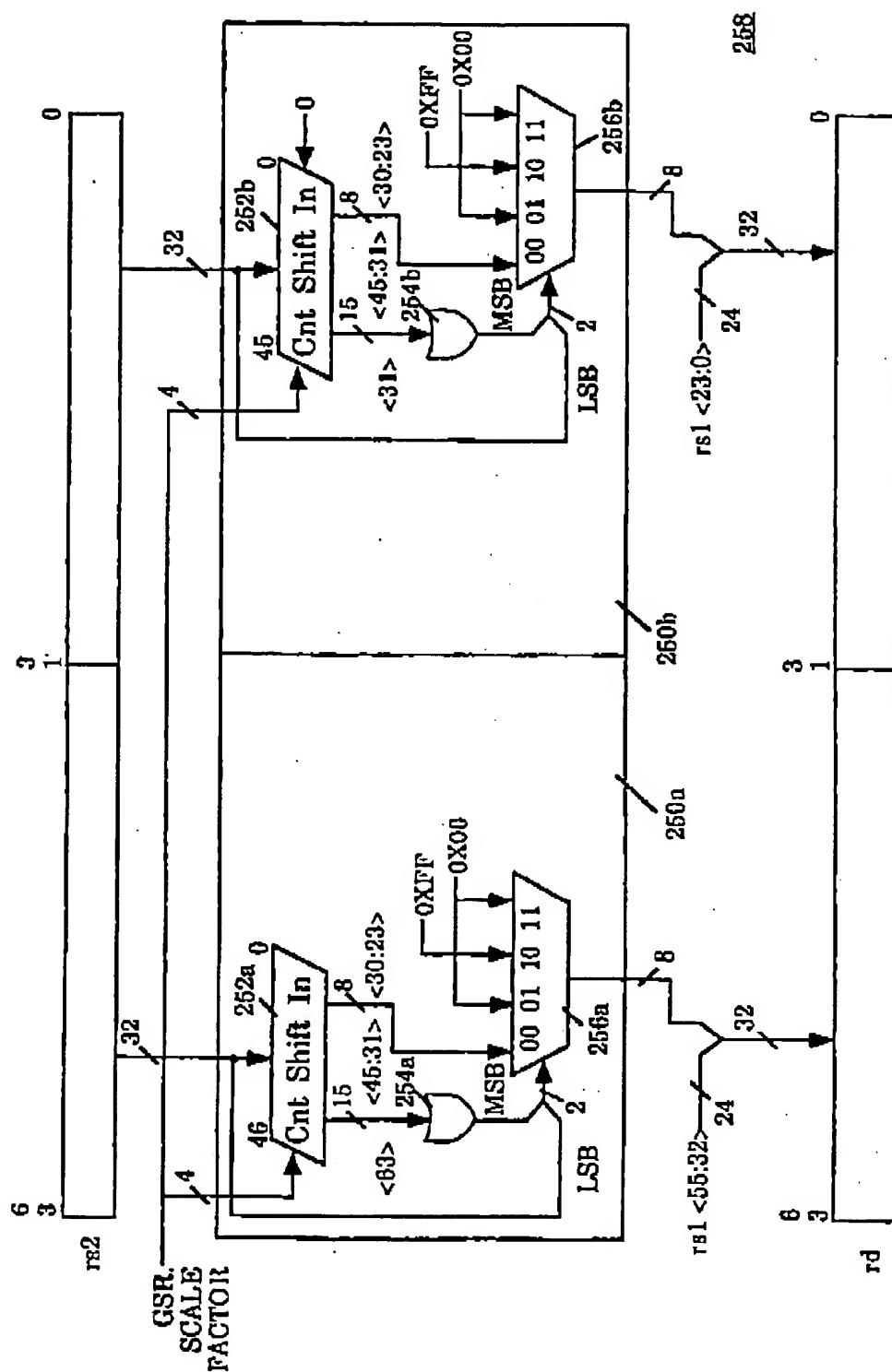


Figure 8f

U.S. Patent

Aug. 17, 1999

Sheet 14 of 20

5,938,756

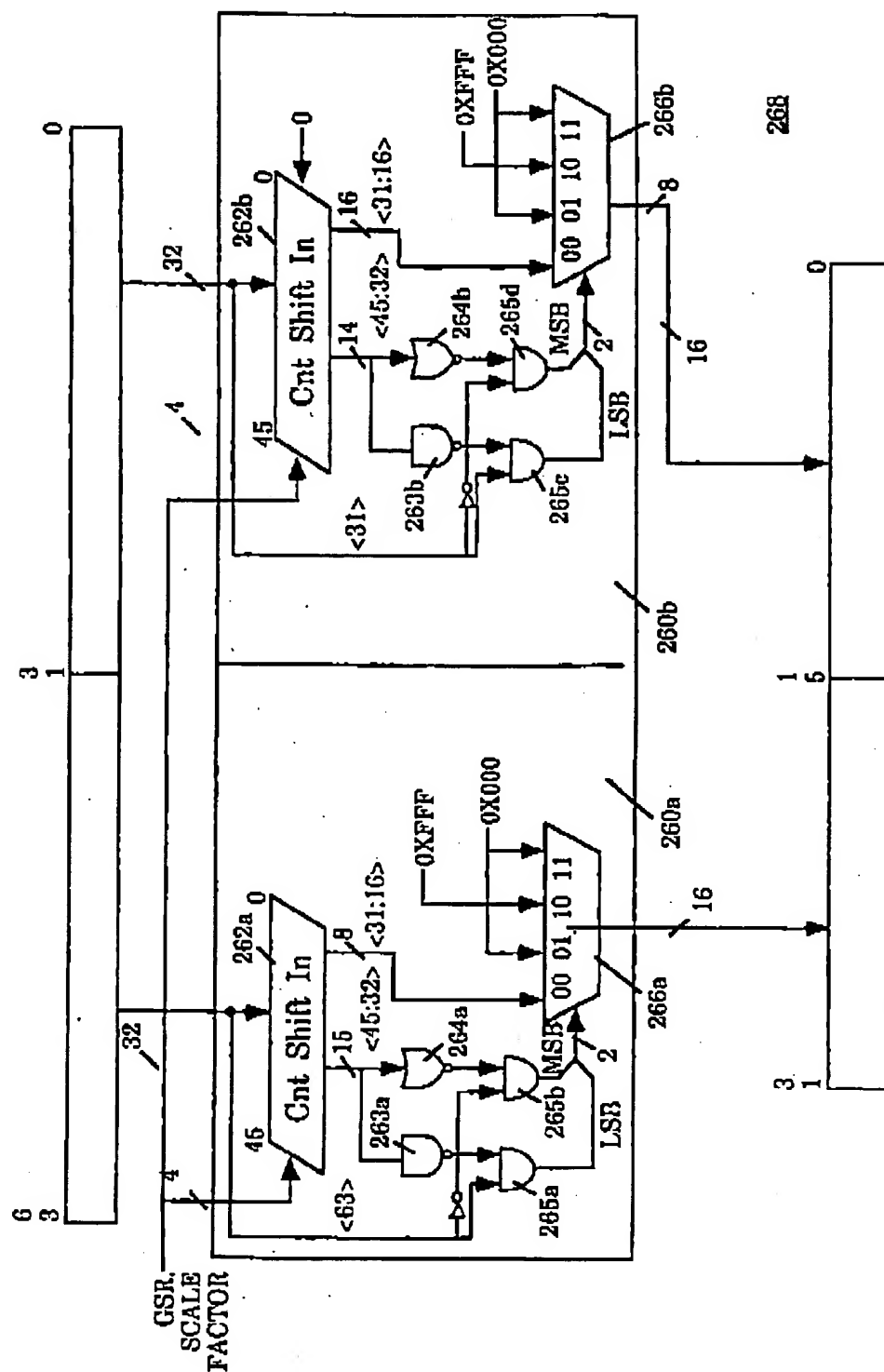


Figure 8g

**U.S. Patent****Aug. 17, 1999****Sheet 15 of 20****5,938,756**

op = 10 , op3 = 110110

opcode	opf	operation
138 - PDIST	000111110	distance between 8 8-bit components

Exemplary Assembly Language Syntax		
pdist	freg <sub>rs1</sub> ,	freg <sub>rs2</sub> , freg <sub>rd</sub>

***Figure 9a***



U.S. Patent

Aug. 17, 1999

Sheet 16 of 20

5,938,756

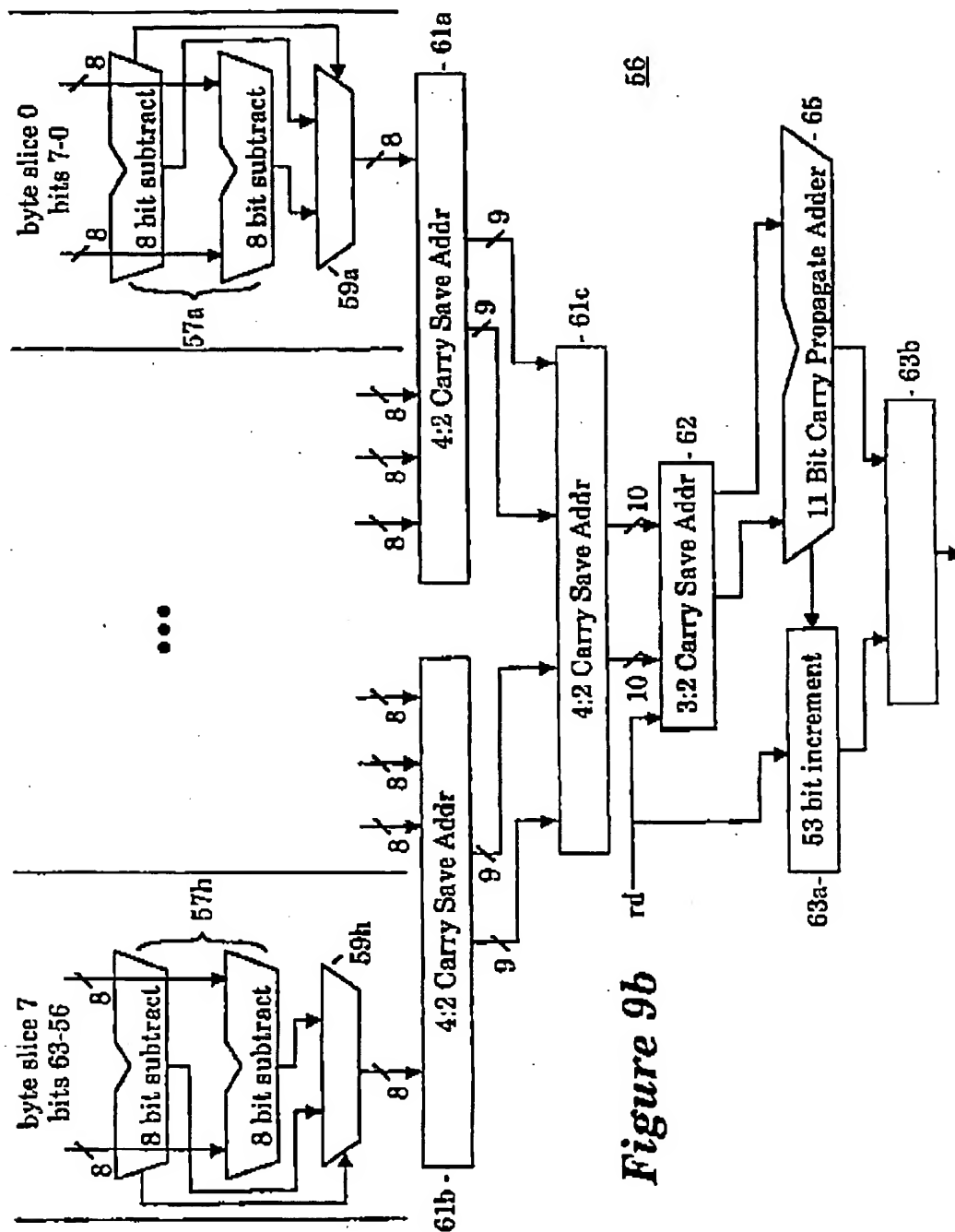


Figure 9b

**U.S. Patent**

Aug. 17, 1999

Sheet 17 of 20

**5,938,756**

op = 10 , op3 = 110110

	opcode	opf	operation
140a	EDGE8	00000000	8 8-bit edge boundary processing
140b	EDGE8L	00000010	8 8-bit edge boundary processing, little endian
140c	EDGE16	00000100	4 16-bit edge boundary processing
140d	EDGE16L	00000110	4 16-bit edge boundary processing, little endian
140e	EDGE32	00001000	2 32-bit edge boundary processing
140f	EDGE32L	00001010	2 32-bit edge boundary processing, little endian

Exemplary Assembly Language Syntax	
edge8	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
edge8l	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
edge16	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
edge16l	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
edge32	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>
edge32l	reg <sub>rs1</sub> , reg <sub>rs2</sub> , reg <sub>rd</sub>

**Figure 10a**

U.S. Patent

Aug. 17, 1999

Sheet 18 of 20

5,938,756

## BIG ENDIAN

Edge Size	LSB	Left Edge	Right Edge
8	000	11111111	10000000
8	001	01111111	11000000
8	010	00111111	11100000
8	011	00011111	11110000
8	100	00001111	11111000
8	101	00000111	11111100
8	110	00000011	11111110
8	111	00000001	11111111
16	00x	1111	1000
16	01x	0111	1100
16	10x	0011	1110
16	11x	0001	1111
32	0xx	11	10
32	1xx	01	11

## LITTLE ENDIAN

Edge Size	LSB	Left Edge	Right Edge
8	000	11111111	00000001
8	001	11111110	00000011
8	010	11111100	00000111
8	011	11111000	00001111
8	100	11110000	00011111
8	101	11100000	00111111
8	110	11000000	01111111
8	111	10000000	11111111
16	00x	1111	0001
16	01x	1110	0011
16	10x	1100	0111
16	11x	1000	1111
32	0xx	11	01
32	1xx	10	11

Figure 10b

U.S. Patent

Aug. 17, 1999

Sheet 19 of 20

5,938,756

op = 10, op3 = 110110

	opcode	opf	operation
142a	ARRAY8	000010010	convert 8-bit 3-D address to blocked byte address
142b	ARRAY16	000010010	convert 16-bit 3-D address to blocked byte address
142c	ARRAY32	000010010	convert 32-bit 3-D address to blocked byte address

rs2 value	number of elements	rs2 value	number of elements
0	64	3	512
1	128	4	1024
2	256	5	2048

rs1:	z integer	z fraction	y integer	y fraction	x integer	x fraction
	63	55 54	44 43	33 32	22 21	11 10 0

8 bits:	upper			middle			lower		
	Z	Y	X	Z	Y	X	Z	Y	X
	20	17	17	17	13	9	5	4	2 0
	+ 2 rs2 + 2 rs2 + rs2								

16 bits:	upper			middle			lower			0
	Z	Y	X	Z	Y	X	Z	Y	X	
	21	18	18	18	14	10	6	5	3	1 0
	+ 2 rs2 + 2 rs2 + rs2									

32 bits:	upper			middle			lower			00
	Z	Y	X	Z	Y	X	Z	Y	X	
	22	19	19	19	15	11	7	6	4	2 0
	+ 2 rs2 + 2 rs2 + rs2									

Exemplary Assembly Language Syntax		
array8	reg <sub>rs1</sub> ,	reg <sub>rs2</sub> , reg <sub>rd</sub>
array16	reg <sub>rs1</sub> ,	reg <sub>rs2</sub> , reg <sub>rd</sub>
array32	reg <sub>rs1</sub> ,	reg <sub>rs2</sub> , reg <sub>rd</sub>

Figure 11a

U.S. Patent

Aug. 17, 1999

Sheet 20 of 20

5,938,756

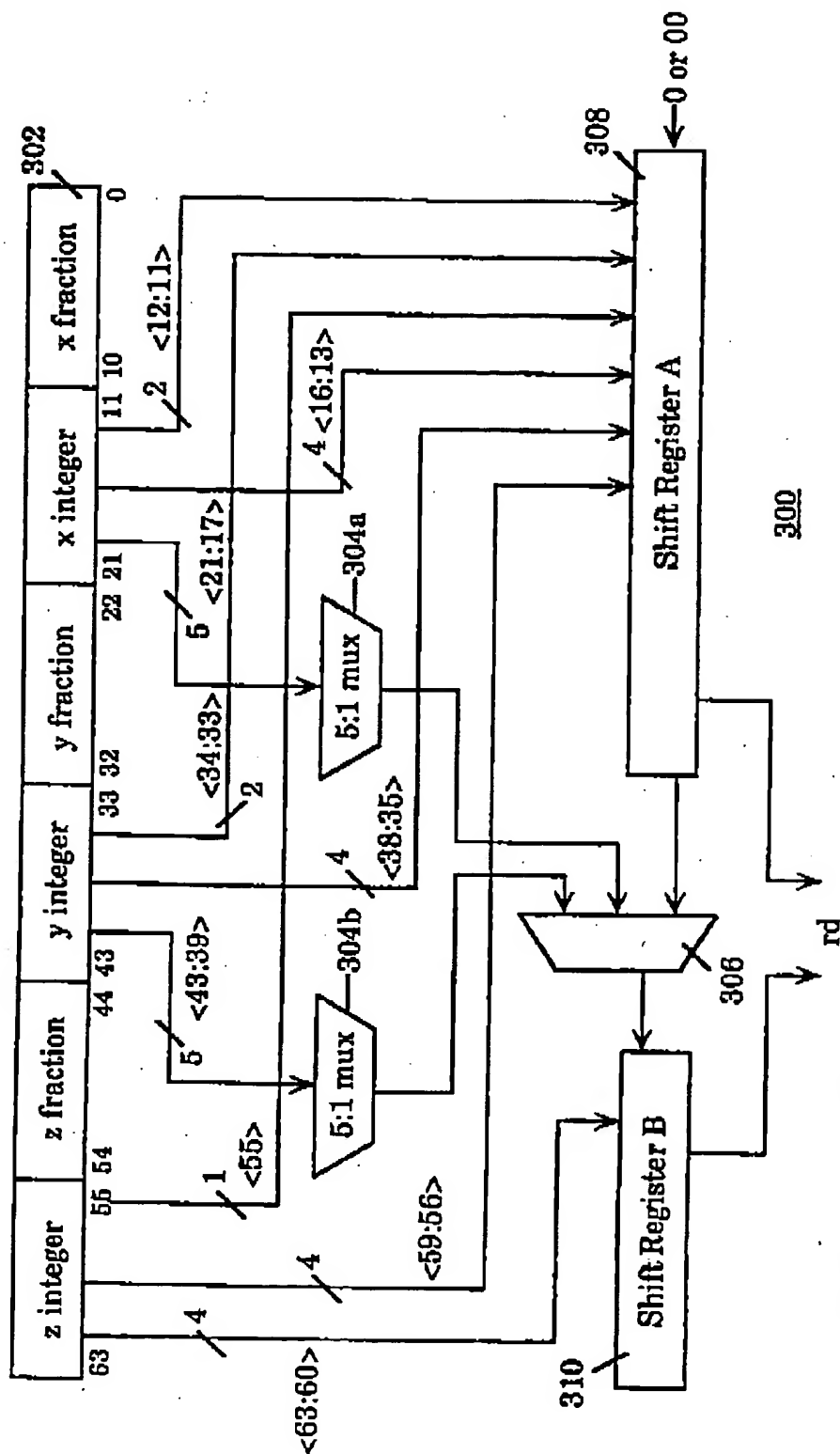


Figure 11b

5,938,756

1

# CENTRAL PROCESSING UNIT WITH INTEGRATED GRAPHICS FUNCTIONS

This is a Continuation of application Ser. No. 08/236,572, filed Apr. 29, 1994 now abandoned.

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention

The present invention relates to the field of computer systems. More specifically, the present invention relates to a cost effective, high performance central processing unit (CPU) having integrated graphics capabilities.

### 2. Background

There are three major barriers to achieving high performance in graphics computer systems. The first barrier is in floating point processing throughput. Graphics applications typically perform large amount of figure manipulation operations such as transformations and clippings using floating point data. The second barrier is in integer or fixed point processing throughput. Graphics applications also typically perform large amount of display operations such as scan conversion and color interpolation using integer or fixed point data. The third barrier is in memory references. The above described operations typically require large amount of memory references for reading from and writing into for example the frame and Z-buffers.

Historically, the CPUs in early prior art computer systems are responsible for both graphics as well as non-graphics functions. No special hardware are provided to assist these early CPUs in performing the large amount of floating and fixed point processing, nor memory references. While the designs of these early prior art computer systems are simple, their performance are typically slow.

Some later prior art computer systems provide auxiliary display processors. The auxiliary display processors would off load these later CPUs from some of the display related operations. However, these later CPUs would still be responsible for most of the graphics processing. Typically, the bandwidth of the system buses of these later prior art computer systems are increased correspondingly to accommodate the increased amount of communications between the processors over the buses. The auxiliary display processors may even be provided with their own memory to reduce the amount of memory contentions between the processors. While generally performance will increase, however, the approach is costly and complex.

Other later prior art computer systems would provide auxiliary graphics processors with even richer graphics functions. The auxiliary graphics processors would off load the CPUs of these later prior art computer systems from most of the graphics processing. Under this approach extensive dedicated hardware as well as sophisticated software interface between the CPUs and the auxiliary graphics processors will have to be provided. While performance will increase even more, however, the approach is even more costly and more complex than the display processor approach.

In the case of microprocessors, as the technology continues to allow more and more circuitry to be packaged in a small area, it is increasingly more desirable to integrate the general purpose CPU with built-in graphics capabilities instead. Some modern prior art computer systems have begun to do that. However, the amount and nature of graphics functions integrated in these modern prior art computer systems typically are still very limited. Particular

2

graphics functions known to have been integrated include only frame buffer checks, add with pixel merge, and add with z-buffer merge. Much of the graphics processing on these modern prior art systems remain being processed by the general purpose CPU without additional built-in graphics capabilities, or by the auxiliary display/graphics processors.

As will be disclosed, the present invention provides a cost effective, high performance CPU with integrated native graphics capabilities that advantageously overcomes much of these performance barriers and achieves the above described and other desired results.

## SUMMARY OF THE INVENTION

Under the present invention, the desired results are advantageously achieved by providing a graphics execution unit (GRU) to the central processing unit (CPU). The GRU comprises a graphics status register (GSR) and at least one partitioned execution path. The GSR is used to store a graphics data scaling factor and a graphics data alignment address offset. The at least one partitioned execution path is used to execute a number of graphics operations on graphics data having a number of graphics data formats. Some of these graphic operations are partitioned operations operating simultaneously on multiple components of graphics data, including graphics operations operating in accordance to the graphics data scaling factor and alignment address offset.

In one embodiment, the GRU comprises a first and a second partitioned execution path. The two execution paths are independent of each other. The first partitioned execution path is used to independently execute a number of partitioned addition and subtraction, expansion, merge, and logical operations on graphics data, and a number of alignment operations on graphics data using the alignment address offset. The second partitioned execution path is used to independently execute a number of partitioned multiplication, a number of pixel distance computation, and compare operations on graphics data, and a number of data packing operations on graphics data using the scaling factor.

Additionally, under this embodiment, the integer execution unit (IEU) of the CPU is used to execute a number of edge handling operations on graphics data addresses, and enhanced with additional circuitry for 3-D array address conversions, while the load and store unit (LSU) of the CPU is also used to execute a number of graphics data load and store operations, including partial conditional store operations.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the CPU of an exemplary graphics computer system incorporating the teachings of the present invention.

FIG. 2 illustrates the relevant portions of one embodiment of the Graphics Execution Unit (GRU) in further detail.

FIG. 3 illustrates the Graphics Status Register (GSR) of the GRU in further detail.

FIG. 4 illustrates the first partitioned execution path of the GRU in further detail.

FIG. 5 illustrates the second partitioned execution path of the GRU in further detail.

FIGS. 6a-6c illustrate the graphics data formats, the graphics instruction formats, and the graphic instruction groups in further detail.

FIGS. 7a-7c illustrate the graphics data alignment instructions and circuitry in further detail.

5,938,756

3

FIGS. 8a-8g illustrate the graphics data packing instructions and circuitry in further detail.

FIGS. 9a-9b illustrate the graphics data pixel distance computation instruction and circuitry in further detail.

FIGS. 10a-10b illustrate the graphics data edge handling instructions in further detail.

FIGS. 11a-11b illustrate the graphics data 3-D array addressing instructions and circuitry in further detail.

#### DETAILED DESCRIPTION

In the following description, for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are shown in diagrammatic or block diagram form in order not to obscure the present invention.

Referring now to FIG. 1, a block diagram illustrating the CPU of an exemplary graphics computer system incorporating the teachings of the present invention is shown. As illustrated, the CPU 24 comprises a prefetch and dispatch unit (PDU) 46 including an instruction cache 40, an integer execution unit (IEU) 30, an integer register file 36, a floating point unit (FPU) 26, a floating point register file 38, and a graphics execution unit (GRU) 28, coupled to each other as shown. Additionally, the CPU 24 comprises two memory management units (MMU & DMMU) 44a-44b, and a load and store unit (LSU) 48 including a data cache 42, coupled to each other and the previously described elements as shown. Together they fetch, dispatch, execute, and save execution results of instructions, including graphics instructions, in a pipelined manner.

The PDU 46 fetches instructions from memory and dispatches them to the IEU 30, the FPU 26, the GRU 28, and the LSU 48 accordingly. Prefetched instructions are stored in the instruction cache 40. The IEU 30, the FPU 26, and the GRU 28 perform integer, floating point, and graphics operations respectively. In general, the integer operands/results are stored in the integer register file 36, whereas the floating point and graphics operands/results are stored in the floating point register file 38. Additionally, the IEU 30 also performs a number of graphics operations, and appends address space identifiers (ASI) to addresses of load/store instructions for the LSU 48, identifying the address spaces being accessed. The LSU 48 generates addresses for all load and store operations. The LSU 48 also supports a number of load and store operations, specifically designed for graphics data. Memory references are made in virtual addresses. The MMUs 44a-44b map virtual addresses to physical addresses.

There are many variations to how the PDU 46, the IEU 30, the FPU 26, the integer and floating point register files 36 and 38, the MMUs 44a-44b, and the LSU 48, are coupled to each other. In some variations, some of these elements 46, 30, 26, 36, 38, 44a-44b, and 48, may be combined, while in other variations, some of these elements 46, 30, 26, 36, 38, 44a-44b, and 48, may perform other functions. Thus, except for the incorporated teachings of the present invention, these elements 46, 30, 26, 36, 38, 44a-44b, and 48, are intended to represent a broad category of PDUs, IEUs, FPUs, integer and floating point register files, MMUs, and LSUs, found in many graphics and non-graphics CPUs. Their constitutions and functions are well known and will not be otherwise described further. The teachings of the present invention incorporated in these elements 46, 30, 26, 36, 38, 44a-44b, and 48, and the GRU 28 will be described in further detail below.

4

Referring now to FIG. 2, a block diagram illustrating the relevant portions of one embodiment of the GRU in further detail is shown. In this embodiment, the GRU 28 comprises a graphics status register (GSR) 50, a first and a second partitioned execution path 32 and 34. The two execution paths 32 and 34 are independent of each other. In other words, two graphics instructions can be independently issued into the two execution paths 32 and 34 at the same time. Together, they independently execute the graphics instructions, operating on graphics data. The functions and constitutions of these elements 50, 32 and 34 will be described in further detail below with additional references to the remaining figures.

Referring now to FIG. 3, a diagram illustrating the relevant portions of one embodiment of the graphics status register (GSR) is shown. In this embodiment, the GSR 50 is used to store the least significant three bits of a pixel address before alignment (alignaddr\_offset) 54, and a scaling factor to be used for pixel formatting (scale\_factor) 52. The alignaddr\_offset 54 is stored in bits GSR[2:0], and the scale\_factor 52 is stored in bits GSR[6:3]. As will be described in more detail below, two special instructions RDASR and WRASR are provided for reading from and writing into the GSR 50. The RDASR and WRASR instructions, and the usage of alignaddr\_offset 54 and scale\_factor 52 will be described in further detail below.

Referring now to FIG. 4, a block diagram illustrating the relevant portions of one embodiment of the first partitioned execution path is shown. The first partitioned execution path 32 comprises a partitioned carry adder 37, a graphics data alignment circuit 39, a graphics data expand/merge circuit 60, and a graphics data logical operation circuit 62, coupled to each other as shown. Additionally, the first partitioned execution path 32 further comprises a couple of registers 35a-35b, and a 4:1 multiplexor 43, coupled to each other and the previously described elements as shown. At each dispatch, the PDU 46 may dispatch either a graphics data partitioned add/subtract instruction, a graphics data alignment instruction, a graphics data expand/merge instruction or a graphics data logical operation to the first partitioned execution path 32. The partitioned carry adder 37 executes the partitioned graphics data add/subtract instructions, and the graphics data alignment circuit 39 executes the graphics data alignment instruction using the alignaddr\_offset stored in the GSR 50. The graphics data expand/merge circuit 60 executes the graphics data merge/expand instructions. The graphics data logical operation circuit 62 executes the graphics data logical operations.

The functions and constitutions of the partitioned carry adder 37 are similar to simple carry adders found in many integer execution units known in the art, except the hardware are replicated multiple times to allow multiple additions/subtractions to be performed simultaneously on different partitioned portions of the operands. Additionally, the carry chain can be broken into two 16-bit chains. Thus, the partitioned carry adder 37 will not be further described.

Similarly, the functions and constitutions of the graphics data expand/merge circuit 60, and the graphics data logical operation circuit 62 are similar to expand/merge and logical operation circuits found in many integer execution units known in the art, except the hardware are replicated multiple times to allow multiple expand/merge and logical operations to be performed simultaneously on different partitioned portions of the operands. Thus, the graphics data expand/merge circuit 60, and the graphics data logical operation circuit 62 will also not be further described.

The graphics data partitioned add/subtract and the graphics data alignment instructions, and the graphics data alignment circuit 39 will be described in further detail below.

5,938,756

5

Referring now to FIG. 5, a block diagram illustrating the relevant portion of one embodiment of the second partitioned execution path in further detail is shown. In this embodiment, the second partitioned execution path 34 comprises a pixel distance computation circuit 36, a partitioned multiplier 58, a graphics data packing circuit 59, and a graphics data compare circuit 64, coupled to each other as shown. Additionally, the second partitioned execution path 34 further comprises a number of registers 55a-55c, a 4:1 multiplexor 53, coupled to each other and the previously described elements as shown. At each dispatch, the PDU 46 may dispatch either a pixel distance computation instruction, a graphics data partitioned multiplication instruction, a graphics data packing instruction, or a graphics data compare instruction to the second partitioned execution path 34. The pixel distance computation circuit 56 executes the pixel distance computation instruction. The partitioned multiplier 58 executes the graphics data partitioned multiplication instructions. The graphics data packing circuit 59 executes the graphics data packing instructions. The graphics data compare circuit 64 executes the graphics data compare instructions.

The functions and constructions of the partitioned multiplier 58, and the graphics data compare circuit 64 are similar to simple multipliers, and compare circuits found in many integer execution units known in the art, except the hardware are replicated multiple times to allow multiple multiplications and comparison operations to be performed simultaneously on different partitioned portions of the operands. Additionally, multiple multiplexors are provided to the partitioned multiplier for rounding, and comparison masks are generated by the comparison circuit 64. Thus, the partitioned multiplier 58, and the graphics data compare circuit 64 will not be further described.

The pixel distance computation, graphics data partitioned multiplication, graphics data pack/expand/merge, graphics data logical operation, and graphics data compare instructions, the pixel distance circuit 56, and the graphics data pack circuit 59 will be described in further detail below.

While the present invention is being described with an embodiment of the GRU 28 having two independent partitioned execution paths, and a particular allocation of graphics instruction execution responsibilities among the execution paths, based on the descriptions to follow, it will be appreciated that the present invention may be practiced with one or more independent partitioned execution paths, and the graphics instruction execution responsibilities allocated in any number of manners.

Referring now to FIGS. 6a-6c, three diagrams illustrating the graphics data formats, the graphics instruction formats, and the graphics instructions are shown. As illustrated in FIG. 6a, the exemplary CPU 24 supports three graphics data formats, an eight bit format (Pixel) 66a, a 16 bit format (Fixed16) 66b, and a 32 bit format (Fixed32) 66c. Thus, four pixel formatted graphics data are stored in a 32-bit word, 66a whereas either four Fixed16 or two Fixed32 formatted graphics data are stored in a 64-bit word 66b or 66c. Image components are stored in either the Pixel or the Fixed16 format 66a or 66b. Intermediate results are stored in either the Fixed16 or the Fixed32 format 66b or 6c. Typically, the intensity values of a pixel of an image, e.g. the alpha, green, blue, and red values (A, G, B, R), are stored in the Pixel format 66a. These intensity values may be stored in band interleaved where the various color components of a point in the image are stored together, or band sequential where all of the values for one component are stored together. The Fixed16 and Fixed32 formats 66b-66c provide enough

6

precision and dynamic range for storing intermediate data computed during filtering and other simple image manipulation operations performed on pixel data. Graphics data format conversions are performed using the graphics data pack, expand, merge, and multiply instructions described below.

As illustrated in FIG. 6b, the CPU 24 supports three graphics instruction formats 68a-68c. Regardless of the instruction format 68a-68c, the two most significant bits [31:30] 70a-70c provide the primary instruction format identification, and bits[24:19] 74a-74c provide the secondary instruction format identification for the graphics instructions. Additionally, bits[29:25] (rd) 72a-72c identify the destination (third source) register of a graphics (block/partial conditional store) instruction, whereas, bits [18:14] (rs1) 76a-76c identify the first source register of the graphics instruction. For the first graphics instruction format 68a, bits[13:5] (opf) and bits[4:0] (rs2) 80 and 82a identify the op codes and the second source register for a graphics instruction of the format. For the second and third graphics instruction formats 68b-68c, bits[13:5] (imm\_asl) and bits [13:0] (simm\_13) may optionally identify the ASI. Lastly, for the second graphics instruction format 68b, bits[4:0] (rs2) further provide the second source register for a graphics instruction of the format (or a mask for a partial conditional store).

As illustrated in FIG. 6c, the CPU 24 supports a number of GSR related instructions 200, a number of partitioned add/subtract/multiplication instructions 202 and 208, a number of graphics data alignment instructions 204, a number of pixel distance computation instructions 206, a number of graphics data pack/expand/merge instructions 210 and 212, a number of graphics data logical and compare instructions 214 and 216, a number of edge handling and 3-D array access instructions 218 and 220, and a number of memory access instructions 222.

The GSR related instructions 200 include a RDASR and a WRASR instruction for reading and writing the alignaddr\_offset and the scale\_factor from and into the GSR 50. The RDASR and WRASR instructions are executed by the IEU 30. The RDASR and WRASR instructions are similar to other CPU control register read/write instructions, thus will not be further described.

The graphics data partitioned add/subtract instructions 202 include four partitioned graphics data addition instructions and four partitioned graphics data subtraction instructions for simultaneously adding and subtracting four 16-bit, two 16-bit, two 32-bit, and one 32-bit graphics data respectively. These instructions add or subtract the corresponding fixed point values in the rs1 and rs2 registers, and correspondingly place the results in the rd register. As described earlier, the graphics data partitioned add/subtract instructions 202 are executed by the partitioned carry adder 37 in the first independent execution path 32 of the GRU 28.

The graphics data partitioned multiplication instructions 208 include seven partitioned graphics data multiplication instructions for simultaneously multiplying either two or four 8-bit graphics data with another two or four corresponding 16-bit graphics data. A FMUL8x16 instruction multiplies four 8-bit graphics data in the rs1 register by four corresponding 16-bit graphics data in the rs2 register. For each product, the upper 16 bits are stored in the corresponding positions of the rd register. A FMUL8x16AU and a FMUL8x16AL instruction multiplies the four 8-bit graphics data in the rs1 register by the upper and the lower halves of the 32-bit graphics data in the rs2 register respectively.



5,938,756

7

Similarly, for each product, the upper 16 bits are stored in the corresponding positions of the rd register.

A FMUL8SU×16 instruction multiplies the four upper 8-bits of the four 16-bit graphics data in the rs1 register by the four corresponding 16-bit graphics data in the rs2 register. Likewise, for each product, the upper 16 bits are stored in the corresponding positions of the rd register. A FMUL8UL×16 instruction multiplies the four lower 8-bits of the four 16-bit graphics data in the rs1 register by the four corresponding 16-bit graphics data in the rs2 register. For each product, the sign extended upper 8 bits are stored in the corresponding positions of the rd register.

A FMULD8SU×16 instruction multiplies the two upper 8-bits of the two 16-bit graphics data in the rs1 register by the two corresponding 16-bit graphics data in the rs2 register. For each product, the 24 bits are appended with 8-bit of zeroes and stored in the corresponding positions of the rd register. A FMULD8UL×16 instruction multiplies the two lower 8-bits of the two 16-bit graphics data in the rs1 register by the two corresponding 16-bit graphics data in the rs2 register. For each product, the 24 bits are sign extended and stored in the corresponding positions of the rd register.

As described earlier, the graphics data partitioned multiplication instructions 208 are executed by the partitioned multiplier 58 in the second independent execution path 34 of the GRU 28.

The graphics data expand and merge instructions 210 include a graphics data expansion instruction, and a graphics data merge instruction, for simultaneously expanding four 8-bit graphics data into four 16-bit graphics data, and interleavingly merging eight 8-bit graphics data into four 16-bit graphics data respectively. A FEXPAND instruction takes four 8-bit graphics data in the rs2 register, left shifts each 8-bit graphics data by 4 bits, and then zero-extends each left shifted graphics data to 16-bits. The results are correspondingly placed in the rd register. A FPMERGE instruction interleavingly merges four 8-bit graphics data from the rs1 register and four 8-bit graphics data from the rs2 register, into a 64 bit graphics datum in the rd register. As described earlier, the graphics data expand and merge instructions 210 are executed by the expand/merge portions of the graphics data expand/merge circuit 60 in the first independent execution path 32 of the GRU 28.

The graphics data logical operation instructions 214 include thirty-two logical operation instructions for performing logical operations on graphics data. Four logical operations are provided for zeroes filling or ones filling the rd register in either single or double precision. Four logical operation instructions are provided for copying the content of either the rs1 or rs2 register into the rd register in either single or double precision. Four logical operation instructions are provided for negating the content of either the rs1 or rs2 register and storing the result into the rd register in either single or double precision. Some logical operations are provided to perform a number of Boolean operations against the content of the rs1 and rs2 registers in either single or double precision, and storing the Boolean results into the rd register. Some of these Boolean operations are performed after having either the content of the rs1 or the rs2 register negated first. As described earlier, these graphics data logical operation instructions 214 are executed by the graphics data logical operation circuit 62 in the first independent execution path 32 of the GRU 28.

The graphics data compare instructions 216 include eight graphics data compare instructions for simultaneously comparing four pairs of 16-bit graphics data or two pairs of

8

32-bit graphics data. The comparisons between the graphics data in the rs1 and rs2 registers include greater than, less than, not equal, and equal. Four or two result bits are stored in the least significant bits in the rd register. Each result bit is set if the corresponding comparison is true. Complimentary comparisons between the graphics data, i.e., less than or equal to, and greater than or equal to, are performed by swapping the graphics data in the rs1 and rs2 registers. As described earlier, these graphics data compare instructions 216 are executed by the graphics data compare circuit 62 in the first independent execution path 32 of the GRU 28.

The graphics data memory reference instructions 222 include a partial (conditional) store, a short load, a short store, a block load and a block store instruction. The graphics data load and store instructions are qualified by the imm\_asi and asi values to determine whether the graphics data load and store instructions 144 and 146 are to be performed simultaneously on 8-bit graphics data, 16-bit graphics data, and whether the operations are directed towards the primary or secondary address spaces in big or little endian format. For the store operations, the imm\_asi and asi values further serve to determine whether the graphics data store operations are conditional.

A partial (conditional) store operation stores the appropriate number of values from the rd register to the addresses specified by the rs1 register using the mask specified (in the rs2 bit location). Mask has the same format as the results generated by the pixel compare instructions. The most significant bit of the mask corresponds to the most significant part of the rs1 register. A short 8-bit load operation may be performed against arbitrary byte addresses. For a short 16-bit load operation, the least significant bit of the address must be zero. Short loads are zero extended to fill the entire floating point destination register. Short stores access either the low order 8 or 16 bits of the floating point source register. A block load/store operation transfers data between 8 contiguous 64-bit floating point registers and an aligned 64-byte block in memory.

As described earlier, these graphics data memory reference instructions 222 are executed by the LSU 48 of the CPU 24.

The graphics data alignment instructions 204, the pixel distance computation instructions 206, the graphics data pack instructions 212, the edge handling instructions 218, and the 3-D array accessing instructions 220 will be described in further detail below in conjunction with the pixel distance computation circuit 56 and the graphics data pack circuit 59 in the second independent execution path 34 of the GRU 28.

Referring now to FIGS. 7a-7c, the graphics data alignment instructions, and the relevant portions of one embodiment of the graphics data alignment circuit are illustrated. As shown in FIG. 7a, there are two graphics data address calculation instructions 98a-98b, and one graphics data alignment instruction 100 for calculating addresses of misaligned graphics data, and aligning misaligned graphics data.

The ALIGNADDR instruction 98a adds the content of the rs1 and rs2 registers, and stores the result, except the least significant 3 bits are forced to zeroes, in the rd register. The least significant 3 bits of the result are stored in the alignaddr\_offset field of GSR 50. The ALIGNADDR instruction 98b is the same as the alignaddr instruction 98a, except two's complement of the least significant 3 bits of the result is stored in the alignaddr\_offset field of GSR 50.

The FALIGNDATA instruction 100 concatenates two 64-bit floating point values in the rs1 and rs2 registers to

5,938,756

9

form a 16-byte value. The floating point value in the rs1 register is used as the upper half of the concatenated value, whereas the floating point value in the rs2 register is used as the lower half of the concatenated value. Bytes in the concatenated value are numbered from the most significant byte to the least significant byte, with the most significant byte being byte 0. Eight bytes are extracted from the concatenated value, where the most significant byte of the extracted value is the byte whose number is specified by the alignaddr\_offset field of GSR 50. The result is stored as a 64 bit floating point value in the rd register.

Thus, as illustrated in FIG. 7b, by using the ALIGNADDRESS {LITTLE} instruction to generate and store the alignaddr\_offset in the GSR 50 (step a), copying the two portions of a misaligned graphics data block 99a-99b from memory into the rs1 and rs2 registers, aligning and storing the aligned graphics data block into the rd register using the FALIGNDATA instruction, and then copying the aligned graphics data block 101 from the rd register into a new memory location, a misaligned graphics data block 99a-99b can be aligned in a quick and efficient manner.

As shown in FIG. 7c, in this embodiment, the graphics data alignment circuit 39 comprises a 64-bit multiplexors 51, coupled to each other and the floating point register file as shown. The multiplexor 51 aligns misaligned graphics data as described above.

Referring now to FIGS. 8a-8g, the graphics data packing instructions, and the relevant portions of the packing portion of the graphics data pack/expand/merge circuit are illustrated. As illustrated in FIGS. 8a-8d, there are three graphics data packing instructions 106a-106c, for simultaneously packing four 16-bit graphics data into four 8-bit graphics data, two 32-bit graphics data into two 8-bit graphics data, and two 32-bit graphics data into two 16-bit graphics data.

The FPACK16 instruction 106a takes four 16-bit fixed values in the rs2 register, left shifts them in accordance to the scale\_factor in GSR 50 and maintaining the clipping information, then extracts and clips 8-bit values starting at the corresponding immediate bits left of the implicit binary positions (between bit 7 and bit 6 of each 16-bit value). If the extracted value is negative (i.e., msb is set), zero is delivered as the clipped value. If the extracted value is greater than 255, 255 is delivered. Otherwise, the extracted value is the final result. The clipped values are correspondingly placed in the rd register.

The FPACK32 instruction 106b takes two 32-bit fixed values in the rs2 register, left shifts them in accordance to the scale\_factor in GSR 50 and maintaining the clipping information, then extracts and clips 8-bit values starting at the immediate bits left of the implicit binary positions (i.e., between bit 23 and bit 22 of a 32-bit value). For each extracted value, clipping is performed in the same manner as described earlier. Additionally, the FPACK32 instruction 106b left shifts each 32-bit value in the rs1 register by 8 bits. Finally, the FPACK32 instruction 106b correspondingly merges the clipped values from the rs2 register with the shifted values from the rs1 register, with the clipped values occupying the least significant byte positions. The resulting values are correspondingly placed in the rd register.

The FPACKFIX instruction 106c takes two 32-bit fixed values in the rs2 register, left shifts each 32-bit value in accordance to the scale\_factor in GSR 50 maintaining the clipping information, then extracts and clips 16-bit values starting at the immediate bits left of the implicit binary positions (i.e., between bit 16 and bit 15 of a 32-bit value). If the extracted value is less than -32768, -32768 is

10

delivered as the clipped value. If the extracted value is greater than 32767, 32767 is delivered. Otherwise, the extracted value is the final result. The clipped values are correspondingly placed in the rd register.

As illustrated in FIGS. 8e-8g, in this embodiment, the graphics data packing circuit 59 comprises circuitry 248, 258 and 268 for executing the FPACK16, FPACK32, and FPACKFIX instructions respectively.

The circuitry 248 for executing the FPACK16 instruction comprises four identical portions 240a-240d, one for each of the four corresponding 16-bit fixed values in the rs2 register. Each portion 240a or 240d comprises a shifter 242a, . . . or 242d, an OR gate 244a, . . . or 244d, and a multiplexor 246a, . . . or 246d, coupled to each other as shown. The shifter 242a, . . . or 242d shifts the corresponding 16-bit fixed value (excluding the sign bit) according to the scale factor stored in the GSR 50. The sign bit and the logical OR of bits [29:15] of each of the shift results are used to control the corresponding multiplexor 246a, . . . or 246d. Either bits [14:7] of the shift result, the value 0xFF or the value 0x00 are output.

The circuitry 258 for executing the FPACK32 instruction comprises two identical portions 250a-250b, one for each of the two corresponding 32-bit fixed values in the rs2 register. Each portion 250a or 250b also comprises a shifter 252a or 252d, an OR gate 254a or 254b, and a multiplexor 256a or 256b, coupled to each other as shown. The shifter 252a or 252d shifts the corresponding 32-bit fixed value (excluding the sign bit) according to the scale factor stored in the GSR 50. The sign bit and the logical OR of bits [45:31] of each of the shift results are used to control the corresponding multiplexor 256a or 256b. Either bits [30:23] of the shift result, the value 0xFF or the value 0x00 are output. The output is further combined with either bits [55:32] or bits [23:0] of the rs1 register.

The circuitry 268 for executing the FPACKFIX instruction also comprises two identical portions 260a-260b, one for each of the two corresponding 32-bit fixed values in the rs2 register. Each portion 260a or 260b also comprises a shifter 262a or 262d, a NAND gate 263a or 263b, a NOR gate 264a or 264b, two AND gates 265a-265b or 265c-265d, and a multiplexor 266a or 266b, coupled to each other as shown. The shifter 262a or 262d shifts the corresponding 32-bit fixed value (excluding the sign bit) according to the scale factor stored in the GSR 50. The logical AND of the sign bit and the logical NAND of bits [45:32] of each of the shift results, and the logical AND of the inverted sign bit and the logical NOR of bits [45:32] of each of the shift results, are used to control the corresponding multiplexor 266a or 266b. Either bits [31:16] of the shift result, the value 0xFFFF or the value 0x8000 are output.

Referring now to FIGS. 9a-9b, the pixel distance computation instructions, and the pixel distance computation circuit are illustrated. As shown in FIG. 9a, there is one graphics data distance computation instruction 138 for simultaneously accumulating the absolute differences between graphics data, eight pairs at a time. The PDIST instruction 138 subtracts eight 8-bit graphics data in the rs1 register from eight corresponding 8-bit graphics data in the rs2 register. The sum of the absolute values of the differences is added to the content of the rd register. The PDIST instruction is typically used for motion estimation in video compression algorithms.

As shown in FIG. 9b, in this embodiment, the pixel distance computation circuit 36 comprises eight pairs of 8 bit subtractors 57a-57h. Additionally, the pixel distance

5,938,756

11

computation circuit 56 further comprises three 4:2 carry save adders 61a-61c, a 3:2 carry save adder 62, two registers 63a-63b, and a 11-bit carry propagate adder 65, coupled to each other as shown. The eight pairs of 8 bit subtractors 57a-57h, the three 4:2 carry save adders 61a-61c, the 3:2 carry save adder 62, the two registers 63a-63b, and the 11-bit carry propagate adder 65, cooperate to compute the absolute differences between eight pairs of 8-bit values, and aggregate the absolute differences into a 64-bit sum.

Referring now to FIGS. 10a-10b, the graphics data edge banding instructions are illustrated. As illustrated, there are six graphics edge handling instructions 140a-140f, for simultaneously generating eight 8-bit edge masks, four 16-bit edge masks, and two 32-bit edge masks in big or little endian format.

The masks are generated in accordance to the graphics data addresses in the rs1 and rs2 registers, where the addresses of the next series of pixels to render and the addresses of the last pixels of the scan line are stored respectively. The generated masks are stored in the least significant bits of the rd register.

Each mask is computed from the left and right edge masks as follows:

- The left edge mask is computed from the 3 least significant bits (LSBs) of the rs1 register, and the right edge mask is computed from the 3 (LSBs) of the rs2 register in accordance to FIG. 10b.
- If 32-bit address masking is disabled, i.e. 64-bit addressing, and the upper 61 bits of the rs1 register are equal to the corresponding bits of the rs2 register, then the rd is set equal to the right edge mask ANDed with the left edge mask.
- If 32-bit address masking is enabled, i.e. 32-bit addressing, and the upper 29 bits ([26:2]) the rs1 register are equal to the corresponding bits of the rs2 register, then the rd register is set to the right edge mask ANDed with the left edge mask.
- Otherwise, rd is set to the left edge mask.

Additionally, a number of condition codes are modified as follows:

- a 32-bit overflow condition code is set if bit 31 (the sign) of rs1 and rs2 registers differ and bit 31 (the sign) of the difference differs from bit 31 (the sign) of rs1; a 64-bit overflow condition code is set if bit 63 (the sign) of rs1 and rs2 registers differ and bit 63 (the sign) of the difference differs from bit 63 (the sign) of rs1.
- a 32-bit negative condition code is set if bit 31 (the sign) of the difference is set; a 64-bit negative condition code is set if bit 63 (the sign) of the difference is set.
- a 32-bit zero condition code is set if the 32-bit difference is zero; a 64-bit zero condition code is set if the 64-bit difference is zero.

As described earlier, the graphics edge handling instructions 140a-140f are executed by the IEU 30. No additional hardware is required by IEU 30.

Referring now to FIGS. 11a-11b, the 3-D array addressing instructions and circuitry are illustrated. As illustrated in FIG. 11a, there are three 3-D array addressing instructions 142a-142c for converting 8-bit, 16-bit, and 32-bit 3-D addresses to blocked byte addresses.

Each of these instructions 142a-142c converts 3-D fixed point addresses in the rs1 register to a blocked byte address, and store the resulting blocked byte address in the rd register. These instructions 142a-142c are typically used for address interpolation for planar reformatting operations.

12

Blocking is performed at the 64-byte level to maximize external cache block reuse, and at the 64k-byte level to maximize the data cache's translation lookaside buffer (TLB) entry reuse, regardless of the orientation of the address interpolation. The element size, i.e., 8-bits, 16-bits, or 32-bit, is implied by the instruction. The value of the rs2 register specifies the power of two sizes of the X and Y dimension of a 3D image array. In the embodiment illustrated, the legal values are from zero to five. A value of zero specifies 64 elements, a value of one specifies 128 elements, and so on up to 2048 elements for the external cache block size specified through the value of five. The integer parts of X, Y, and Z (rs1) are converted to either the 8-bit, the 16-bit, or the 32-bit format. The bits above Z upper are set to zero. The number of zeros in the least significant bits is determined by the element size. An element size of eight bits has no zero, an element size of 16-bits has one zero, and an element size of 32-bits has two zeros. Bits in X and Y above the size specified by the rs2 register is ignored.

As described earlier, the 3-D array addressing instructions 142a-142c are also executed by the IEU 30. FIG. 11b illustrates one embodiment of the additional circuitry provided to the IEU 30. The additional circuitry 300 comprises two shift registers 308 and 310, and a number of multiplexers 304a-304b and 306, coupled to each other as shown. The appropriate bits from the lower and middle integer portions of X, Y, and Z (i.e. bits <12:11>, <34:33>, <55>, <16:13>, <38:35>, and <59:56>) are first stored into shift register A 308. Similarly, the appropriate bits of the upper integer portion of Z (i.e. <63:60>) are stored into shift register B 310. Then, selected bits of the upper integer portions of Y and X are shifted into shift register B 310 in order, depending on the value of rs2. Finally, zero, one, or two zero bits are shifted into shift register A 308, with the shift out bits shifted into shift register B 310, depending on the array element size (i.e. 8, 16, or 32 bits).

While the present invention has been described in terms of presently preferred and alternate embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The method and apparatus of the present invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative of, and not limiting the scope of the present invention.

What is claimed is:

1. A microprocessor comprising:

an instruction fetch and dispatch unit;

at least two pipelined execution units connected in parallel to said instruction fetch and dispatch unit, including integer execution logic, floating point execution logic, and graphics execution logic;

a first register file coupled solely to said integer execution logic and storing integer operands and results of operations performed in said integer execution logic; and

a second register file coupled solely to said floating point execution logic and said graphics execution logic, said second register file storing floating point and graphics operands and results of operations performed in said floating point and graphics execution logic;

wherein said graphics execution logic comprises first and second graphics execution units, each separately coupled to said instruction fetch and dispatch unit;

5,938,756

13

wherein said first graphics execution unit includes an ALU, and said second graphics execution unit includes a multiplier;

wherein said second graphics execution unit further includes a pixel distance computation circuit configured to calculate and accumulate the difference between multiple pairs of pixels, said pixel distance computation circuit and said multiplier being configured in parallel such that only one can receive a decoded instruction from said fetch and dispatch unit in a given clock cycle.

2. The microprocessor of claim 1 wherein said pixel distance computation circuit comprises:

a subtractor configured to subtract multiple pixel values in parallel; and

a plurality of adders for providing a total absolute value sum of the subtraction operations on said multiple pixels.

3. A microprocessor comprising:

an instruction fetch and dispatch unit;

at least two pipelined execution units connected in parallel to said instruction fetch and dispatch unit, including

integer execution logic,  
floating point execution logic, and  
graphics execution logic;

a first register file coupled solely to said integer execution logic and storing integer operands and results of operations performed in said integer execution logic; and

a second register file coupled solely to said floating point execution logic and said graphics execution logic, said second register file storing floating point and graphics operands and results of operations performed in said floating point and graphics execution logic;

wherein said graphics execution logic comprises first and second graphics execution units, each separately coupled to said instruction fetch and dispatch unit;

wherein said first graphics execution unit includes an ALU, and said second graphics execution unit includes a multiplier;

wherein said second graphics execution unit further includes a pixel packing circuit configured to pack N-bit pixels into an M-bit format, where M is less than N, said pixel packing circuit being in parallel with said multiplier.

4. A microprocessor comprising:

an instruction fetch and dispatch unit;

at least two pipelined execution units connected in parallel to said instruction fetch and dispatch unit, including

integer execution logic,  
floating point execution logic, and  
graphics execution logic;

a first register file coupled solely to said integer execution logic and storing integer operands and results of operations performed in said integer execution logic; and

14

a second register file coupled solely to said floating point execution logic and said graphics execution logic, said second register file storing floating point and graphics operands and results of operations performed in said floating point and graphics execution logic;

wherein said graphics execution logic comprises first and second graphics execution units, each separately coupled to said instruction fetch and dispatch unit;

wherein said first graphics execution unit includes an ALU, and said second graphics execution unit includes a multiplier;

wherein said first graphics execution unit further includes a graphics alignment circuit in parallel with said ALU.

5. The microprocessor of claim 4 further comprising a graphics status register accessible by said first and second graphics execution units, and wherein said graphics alignment circuit comprises a multiplexer having inputs coupled to first and second registers in said floating point register files, and a control selection input circuit coupled to said graphics status register.

6. A microprocessor comprising:

an instruction fetch and dispatch unit;

at least two pipelined execution units connected in parallel to said instruction fetch and dispatch unit, including

integer execution logic,  
floating point execution logic, and  
graphics execution logic;

a first register file coupled solely to said integer execution logic and storing integer operands and results of operations performed in said integer execution logic; and

a second register file coupled solely to said floating point execution logic and said graphics execution logic, said second register file storing floating point and graphics operands and results of operations performed in said floating point and graphics execution logic;

wherein said microprocessor includes a cache memory, and said integer execution unit further comprises a dedicated block address conversion circuit, distinct from other integer operation circuitry, for converting pixel addresses from a 3D format having X, Y, and Z coordinates linearly set forth in an address to a blocked byte format having addresses with a less significant portion of said X, Y and Z coordinates followed by a more significant portion of said X, Y and Z coordinates.

7. The microprocessor of claim 6 wherein said blocked byte format further comprises a most significant portion of said X, Y and Z coordinates following said more significant portions, such that said blocked byte address consists of a low, middle and high portion of the X, Y and Z coordinates.

8. The microprocessor of claim 7 wherein said low portion corresponds to a cache line.

9. The microprocessor of claim 7 wherein all addresses specified by said middle portion correspond to a single page of an address for said microprocessor.

\* \* \* \* \*

X. Related Proceedings Appendix: Copies of Decisions Rendered by a Court or the Board in any Prior and Pending Appeals, Interferences or Judicial Proceedings

There are no related appeals or interferences to appellant's knowledge that would have a bearing on any decision of the Board of Patent Appeals and Interferences.

42390P5943C

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☒ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**